

VISIT...

LANZAROTE
Caliente.COM

Нечеткая логика и искусственные нейронные сети



ВВЕДЕНИЕ

Как известно, аппарат нечетких множеств и нечеткой логики уже давно (более 10 лет) с успехом применяется для решения задач, в которых исходные данные являются ненадежными и слабо формализованными. Сильные стороны такого подхода:

- описание условий и метода решения задачи на языке, близком к естественному;

- универсальность: согласно знаменитой теореме FAT (Fuzzy Approximation Theorem), доказанной Б. Коско (B. Kosko) в 1993 г., любая математическая система может быть аппроксимирована системой, основанной на нечеткой логике;

- эффе́ктивность (связана с универсальностью), поясняемая рядом теорем, аналогичных теоремам о полноте для искусственных нейронных сетей, например, теоремой вида: для каждой вещественной непрерывной функции g , заданной на компакте U и для произвольного $\varepsilon > 0$ существует нечеткая экспертная система, формирующая выходную функцию $f(\mathbf{x})$ такую, что $\sup_{\mathbf{x} \in U} \|g(\mathbf{x}) - f(\mathbf{x})\| \leq \varepsilon$, где $\|\cdot\|$ — символ принятого расстояния между функциями.

Вместе с тем для нечетких экспертных и управляющих систем характерны и определенные недостатки:

- 1) исходный набор постулируемых нечетких правил формулируется экспертом-человеком и может оказаться неполным или противоречивым;

- 2) вид и параметры функций принадлежности, описывающих входные и выходные переменные системы, выбираются субъективно и могут оказаться не вполне отражающими реальную действительность.

Для устранения, по крайней мере, частично, указанных недостатков рядом авторов было предложено выполнять нечеткие экспертные и управляющие системы адаптивными — корректируя, по мере работы системы, и правила и параметры функций принадлежности. Среди нескольких вариантов такой адаптации одним из самых удачных, по-видимому, является метод так называемых гибридных нейронных сетей.

Гибридная нейронная сеть формально по структуре идентична многослойной нейронной сети с обучением, например, по алгоритму обратного распространения ошибки, но скрытые слои в ней соответствуют этапам функционирования нечеткой системы. Так:

- 1-й слой нейронов выполняет функцию введения нечеткости на основе заданных функций принадлежности входов;

- 2-й слой отображает совокупность нечетких правил;

- 3-й слой выполняет функцию приведения к четкости.

Каждый из этих слоев характеризуется набором параметров (параметрами функций принадлежности, нечетких решающих правил, активационных функций, весами связей), настройка которых производится, в сущности, так же, как для обычных нейронных сетей.

В книге рассмотрены теоретические аспекты составляющих подобных сетей, именно, аппарат нечеткой логики, основы теории искусственных нейронных сетей и собственно гибридных сетей применительно к задачам управления и принятия решений в условиях неопределенности.

Особое внимание уделено программной реализации моделей указанных подходов инструментальными средствами математической системы MATLAB 5.2/5.3.

Глава 1

НЕЧЕТКАЯ ИНФОРМАЦИЯ И ВЫВОДЫ

Пожалуй, наиболее поразительным свойством человеческого интеллекта является способность принимать правильные решения в обстановке неполной и нечеткой информации. Построение моделей приближенных рассуждений человека и использование их в компьютерных системах будущих поколений представляет сегодня одну из важнейших проблем науки.

Значительное продвижение в этом направлении сделано 30 лет тому назад профессором Калифорнийского университета (Беркли) Лотфи А. Заде (Lotfi A. Zadeh). Его работа «Fuzzy Sets», появившаяся в 1965 г. в журнале *Information and Control*, № 8, заложила основы моделирования интеллектуальной деятельности человека и явилась начальным толчком к развитию новой математической теории.

Л. Заде расширил классическое канторовское понятие множества, допустив, что характеристическая функция (функция принадлежности элемента множеству) может принимать любые значения в интервале $[0; 1]$, а не только значения 0 либо 1. Такие множества были названы им нечеткими (fuzzy). Он определил также ряд операций над нечеткими множествами и предложил обобщение известных методов логического вывода *modus ponens* и *modus tollens*.

Вводя затем понятие *лингвистической переменной* и допустив, что в качестве ее значений (термов) выступают нечеткие множества, Л. Заде создал аппарат для описания процессов интеллектуальной деятельности, включая нечеткость и неопределенность выражений.

Дальнейшие работы профессора Л. Заде и его последователей заложили прочный фундамент новой теории и создали предпосылки для внедрения методов нечеткого управления в инженерную практику.

Уже к 1990 г. по этой проблематике опубликовано свыше 10000 работ, а число исследователей достигло 10000, причем в США,

Европе и СССР по 200–300 человек, около 1000 — в Японии, 2000–3000 — в Индии и около 5000 исследователей в Китае.

В последние 5–7 лет началось использование новых методов и моделей в промышленности и в военном деле. Спектр приложений их широк: от управления процессом отправления и остановки поезда метрополитена, управления грузовыми лифтами и доменной печью до стиральных машин, пылесосов и СВЧ-печей. При этом нечеткие системы позволяют повысить качество продукции при уменьшении ресурсо- и энергозатрат и обеспечивают более высокую устойчивость к воздействию мешающих факторов по сравнению с традиционными системами автоматического управления.

Другими словами, новые подходы позволяют расширить сферу приложения систем автоматизации за пределы применимости классической теории. В этом плане любопытна точка зрения Л. Заде: «Я считаю, что излишнее стремление к точности стало оказывать действие, сводящее на нет теорию управления и теорию систем, так как оно приводит к тому, что исследования в этой области сосредотачиваются на тех и только тех проблемах, которые поддаются точному решению. В результате многие классы важных проблем, в которых данные, цели и ограничения являются слишком сложными или плохо определенными для того, чтобы допустить точный математический анализ, оставались и остаются в стороне по той причине, что они не поддаются математической трактовке. Для того чтобы сказать что-либо существенное для проблем подобного рода, мы должны отказаться от наших требований точности и допустить результаты, которые являются несколько размытыми или неопределенными».

Смещение центра исследований нечетких систем в сторону практических приложений привело к постановке целого ряда проблем, таких как новые архитектуры компьютеров для нечетких вычислений, элементная база нечетких компьютеров и контроллеров, инструментальные средства разработки, инженерные методы расчета и разработки нечетких систем управления и многое другое.

Математическая теория нечетких множеств позволяет описывать нечеткие понятия и знания, оперировать этими знаниями и делать нечеткие выводы.

Нечеткое управление оказывается особенно полезным, когда технологические процессы являются слишком сложными для анализа с помощью общепринятых количественных методов или когда доступные источники информации интерпретируются качественно, неточно или неопределенно. Нечеткая логика, на которой основано нечеткое управление, ближе по духу к человеческому мышлению и естественным языкам, чем традиционные логиче-

ские системы. Нечеткая логика, в основном, обеспечивает эффективные средства отображения неопределенностей и неточностей реального мира. Наличие математических средств отражения нечеткости исходной информации позволяет построить модель, адекватную реальности.

1.1. Нечеткие множества

Пусть E — универсальное множество, x — элемент E , а R — некоторое свойство. Обычное (четкое) подмножество A универсального множества E , элементы которого удовлетворяют свойству R , определяется как множество упорядоченных пар

$$A = \{\mu_A(x)/x\},$$

где $\mu_A(x)$ — *характеристическая функция*, принимающая значение 1, если x удовлетворяет свойству R , и 0 — в противном случае.

Нечеткое подмножество отличается от обычного тем, что для элементов x из E нет однозначного ответа «да–нет» относительно свойства R . В связи с этим нечеткое подмножество A универсального множества E определяется как множество упорядоченных пар

$$A = \{\mu_A(x)/x\},$$

где $\mu_A(x)$ — *характеристическая функция принадлежности* (или просто *функция принадлежности*), принимающая значения в некотором вполне упорядоченном множестве M (например, $M = [0, 1]$).

Функция принадлежности указывает степень (или уровень) принадлежности элемента x подмножеству A . Множество M называют множеством принадлежностей. Если $M = \{0, 1\}$, то нечеткое подмножество A может рассматриваться как обычное или четкое множество.

Примеры записи нечеткого множества

Пусть $E = \{x_1, x_2, x_3, x_4, x_5\}$, $M = [0, 1]$; A — нечеткое множество, для которого

$$\mu_A(x_1) = 0,3; \quad \mu_A(x_2) = 0; \quad \mu_A(x_3) = 1; \quad \mu_A(x_4) = 0,5; \quad \mu_A(x_5) = 0,9.$$

Тогда A можно представить в виде

$$A = \{0,3/x_1; 0/x_2; 1/x_3; 0,5/x_4; 0,9/x_5\},$$

или

$$A = \{0,3/x_1 + 0/x_2 + 1/x_3 + 0,5/x_4 + 0,9/x_5\},$$

или

$$A = \frac{x_1}{0,3} \mid \frac{x_2}{0} \mid \frac{x_3}{1} \mid \frac{x_4}{0,5} \mid \frac{x_5}{0,9}.$$

Замечание. Здесь знак «+» не является обозначением операции сложения, а имеет смысл объединения.

1.1.1. Основные характеристики нечетких множеств. Пусть $M = [0, 1]$ и A — нечеткое множество с элементами из универсального множества E и множеством принадлежностей M .

- Величина $\sup_{x \in E} \mu_A(x)$ называется *высотой* нечеткого множества A . Нечеткое множество A *нормально*, если его высота равна 1, т.е. верхняя граница его функции принадлежности равна 1 ($\sup_{x \in E} \mu_A(x) = 1$). При $\sup_{x \in E} \mu_A(x) < 1$ нечеткое множество называется *субнормальным*.

- Нечеткое множество *пусто*, если $\forall x \in E \mu_A(x) = 0$. Непустое субнормальное множество можно нормализовать по формуле

$$\mu_A(x) := \frac{\mu_A(x)}{\sup_{x \in E} \mu_A(x)}.$$

- Нечеткое множество *унимодально*, если $\mu_A(x) = 1$ только на одном x из E .

- *Носителем* нечеткого множества A является обычное подмножество со свойством $\mu_A(x) > 0$, т.е. *носитель* $A = \{x/x \in E, \mu_A(x) > 0\}$.

- Элементы $x \in E$, для которых $\mu_A(x) = 0,5$, называются *точками перехода* множества A .

Примеры нечетких множеств

1. Пусть $E = \{0, 1, 2, \dots, 10\}$, $M = [0, 1]$. Нечеткое множество «Несколько» можно определить следующим образом: «Несколько» $= 0,5/3 + 0,8/4 + 1/5 + 1/6 + 0,8/7 + 0,5/8$; его характеристики: *высота* $= 1$, *носитель* $= \{3, 4, 5, 6, 7, 8\}$, *точки перехода* $= \{3, 8\}$.

2. Пусть $E = \{0, 1, 2, 3, \dots, n, \dots\}$. Нечеткое множество «Малый» можно определить:

$$\text{«Малый»} = \left\{ \mu_{\text{Малый}}(n) = \frac{1}{1 + (n/10)^2/n} \right\}.$$

3. Пусть $E = \{1, 2, 3, \dots, 100\}$ и соответствует понятию «Возраст», тогда нечеткое множество «Молодой» может быть определено с помощью

$$\mu_{\text{Молодой}}(x) = \begin{cases} 1, & x \in [1, 25], \\ \frac{1}{1 + ((x - 25)/5)^2}, & x > 25. \end{cases}$$

Нечеткое множество «Молодой» на универсальном множестве $E' = \{\text{ИВАНОВ, ПЕТРОВ, СИДОРОВ, ...}\}$ задается с помощью функции принадлежности $\mu_{\text{Молодой}}(x)$ на $E = \{1, 2, 3, \dots, 100\}$ (возраст), называемой по отношению к E' функцией совместимости, при этом:

$$\mu_{\text{Молодой}}(\text{СИДОРОВ}) := \mu_{\text{Молодой}}(x),$$

где x — возраст СИДОРОВА.

4. Пусть $E = \{\text{ЗАПОРОЖЕЦ, ЖИГУЛИ, МЕРСЕДЕС, ...}\}$ — множество марок автомобилей, а $E' = [0, \infty)$ — универсальное множество «Стоимость», тогда на E' мы можем определить нечеткие множества типа:

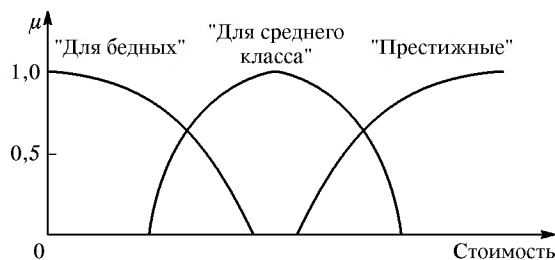


Рис. 1.1. Примеры функций принадлежности

«Для бедных», «Для среднего класса», «Престижные», с функциями принадлежности вида рис. 1.1.

Имея эти функции и зная стоимости автомобилей из E в данный момент времени, мы тем самым определим на E' нечеткие множества с этими же названиями.

Так, например, нечеткое множество «Для бедных», заданное на универсальном множестве $E = \{\text{ЗАПОРОЖЕЦ, ЖИГУЛИ, МЕРСЕДЕС, ...}\}$, выглядит так, как показано на рис. 1.2.

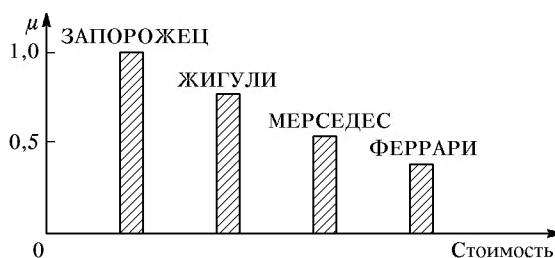


Рис. 1.2. Пример задания нечеткого множества

Аналогично можно определить нечеткое множество «Скоростные», «Средние», «Тихоходные» и т. д.

5. Пусть E — множество целых чисел:

$$E = \{-8, -5, -3, 0, 1, 2, 4, 6, 9\}.$$

Тогда нечеткое подмножество чисел, по абсолютной величине близких к нулю, можно определить, например, так:

$$A = \{0/-8+0,5/-5+0,6/-3+1/0+0,9/1+0,8/2+0,6/4+0,3/6+0/9\}.$$

1.1.2. О методах построения функций принадлежности нечетких множеств. В приведенных выше примерах использованы *прямые* методы, когда эксперт либо просто задает для каждого $x \in E$ значение $\mu_A(x)$, либо определяет функцию совместимости. Как правило, прямые методы задания функции принадлежности используются для измеримых понятий, таких как скорость, время, расстояние, давление, температура и т. д., или когда выделяются полярные значения.

Во многих задачах при характеристике объекта можно выделить набор признаков и для каждого из них определить полярные значения, соответствующие значениям функции принадлежности, 0 или 1.

Например, в задаче распознавания лиц можно выделить шкалы, приведенные в табл. 1.1.

Таблица 1.1. Шкалы в задаче распознавания лиц

		0	1
x_1	высота лба	низкий	высокий
x_2	профиль носа	курносый	горбатый
x_3	длина носа	короткий	длинный
x_4	разрез глаз	узкие	широкие
x_5	цвет глаз	светлые	темные
x_6	форма подбородка	остроконечный	квадратный
x_7	толщина губ	тонкие	толстые
x_8	цвет лица	темный	светлый
x_9	очертание лица	овальное	квадратное

Для конкретного лица A эксперт, исходя из приведенной шкалы, задает $\mu_A(x) \in [0, 1]$, формируя векторную функцию принадлежности $\{\mu_A(x_1), \mu_A(x_2), \dots, \mu_A(x_9)\}$.

При прямых методах используются также групповые прямые методы, когда, например, группе экспертов предъявляют конкретное лицо и каждый должен дать один из двух ответов: «этот человек лысый» или «этот человек не лысый», тогда количество утвердительных ответов, деленное на общее число экспертов, дает

значение $\mu_{\text{лысый}}$ (данного лица). (В этом примере можно действовать через функцию совместимости, но тогда придется считать число волосинок на голове у каждого из предъявленных экспертов лиц.)

Косвенные методы определения значений функции принадлежности используются в случаях, когда нет элементарных измеримых свойств, через которые определяется интересующее нас нечеткое множество. Как правило, это методы попарных сравнений. Если бы значения функций принадлежности были нам известны, например, $\mu_A(x_i) = w_i$, $i = 1, 2, \dots, n$, то попарные сравнения можно представить матрицей отношений $\mathbf{A} = \{a_{ij}\}$, где $a_{ij} = w_i/w_j$ (операция деления).

На практике эксперт сам формирует матрицу $\mathbf{A}$, при этом предполагается, что диагональные элементы равны 1, а для элементов симметричных относительно диагонали $a_{ij} = 1/a_{ji}$, т.е. если один элемент оценивается в α раз сильнее, чем другой, то этот последний должен быть в $1/\alpha$ раз сильнее, чем первый. В общем случае задача сводится к поиску вектора $\mathbf{w}$, удовлетворяющего уравнению вида $\mathbf{A}\mathbf{w} = \lambda_{\max}\mathbf{w}$, где $\lambda_{\max}$ — наибольшее собственное значение матрицы $\mathbf{A}$. Поскольку матрица $\mathbf{A}$ положительна по построению, решение данной задачи существует и является положительным.

Можно отметить еще два подхода:

- *использование типовых форм* кривых для задания функций принадлежности (в форме (L–R)-типа — см. ниже) с уточнением их параметров в соответствии с данными эксперимента;
- *использование относительных частот* по данным эксперимента в качестве значений принадлежности.

1.2. Операции над нечеткими множествами

1.2.1. Логические операции

Включение. Пусть A и B — нечеткие множества на универсальном множестве E . Говорят, что A содержится в B , если $\forall x \in E \mu_A(x) \leq \mu_B(x)$.

Обозначение: $A \subset B$.

Иногда используют термин *доминирование*, т.е. в случае, когда $A \subset B$, говорят, что B доминирует A .

Равенство. A и B равны, если $\forall x \in E \mu_A(x) = \mu_B(x)$.

Обозначение: $A = B$.

Дополнение. Пусть $M = [0, 1]$, A и B — нечеткие множества, заданные на E . A и B дополняют друг друга, если $\forall x \in E \mu_A(x) = 1 - \mu_B(x)$.

Обозначение: $B = \overline{A}$ или $A = \overline{B}$.

Очевидно, что $\overline{\overline{A}} = A$ (дополнение определено для $M = [0, 1]$, но очевидно, что его можно определить для любого упорядоченного M).

Пересечение. $A \cap B$ — наибольшее нечеткое подмножество, содержащееся одновременно в A и B :

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)).$$

Объединение. $A \cup B$ — наименьшее нечеткое подмножество, включающее как A , так и B , с функцией принадлежности:

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)).$$

Разность. $A - B = A \cap \overline{B}$ с функцией принадлежности:

$$\mu_{A-B}(x) = \mu_{A \cap \overline{B}}(x) = \min(\mu_A(x), 1 - \mu_B(x)).$$

Дизъюнктивная сумма

$$A \oplus B = (A - B) \cup (B - A) = (A \cap \overline{B}) \cup (\overline{A} \cap B)$$

с функцией принадлежности:

$$\mu_{A-B}(x) = \max(\min(\mu_A(x), 1 - \mu_B(x)); \min(1 - \mu_A(x), \mu_B(x))).$$

Примеры. Пусть

$$A = 0,4/x_1 + 0,2/x_2 + 0/x_3 + 1/x_4;$$

$$B = 0,7/x_1 + 0,9/x_2 + 0,1/x_3 + 1/x_4;$$

$$C = 0,1/x_1 + 1/x_2 + 0,2/x_3 + 0,9/x_4.$$

Здесь:

1) $A \subset B$, т.е. A содержится в B или B доминирует A ; C не сравнимо ни с A , ни с B , т.е. пары $\{A, C\}$ и $\{A, B\}$ — пары недоминируемых нечетких множеств.

2) $A \neq B \neq C$.

3) $\overline{A} = 0,6/x_1 + 0,8/x_2 + 1/x_3 + 0/x_4$; $\overline{B} = 0,3/x_1 + 0,1/x_2 + 0,9/x_3 + 0/x_4$.

4) $A \cap B = 0,4/x_1 + 0,2/x_2 + 0/x_3 + 1/x_4$.

5) $A \cup B = 0,7/x_1 + 0,9/x_2 + 0,1/x_3 + 1/x_4$.

6) $A - B = A \cap \overline{B} = 0,3/x_1 + 0,1/x_2 + 0/x_3 + 0/x_4$; $B - A = \overline{A} \cap B = 0,6/x_1 + 0,8/x_2 + 0,1/x_3 + 0/x_4$.

7) $A \oplus B = 0,6/x_1 + 0,8/x_2 + 0,1/x_3 + 0/x_4$.

Наглядное представление логических операций над нечеткими множествами. Для нечетких множеств можно строить визуальное представление. Рассмотрим прямоугольную систему координат, на оси ординат которой откладываются значения $\mu_A(x)$, на оси абсцисс в произвольном порядке расположены элементы E (мы уже использовали такое представление в примерах нечетких множеств). Если E по своей природе упорядочено, то этот порядок желательно сохранить в расположении

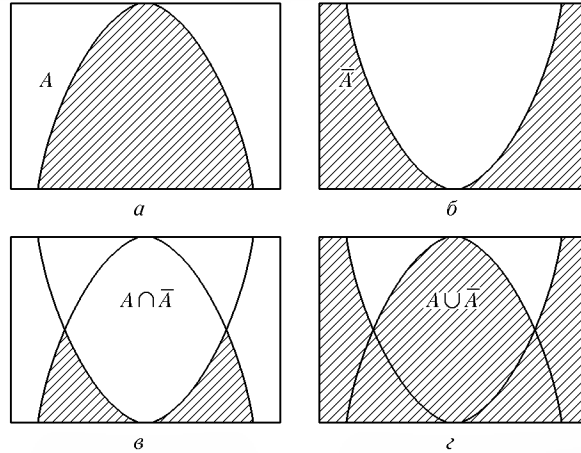


Рис. 1.3. Графическая интерпретация логических операций: a — нечеткое множество A ; $б$ — нечеткое множество $\bar{A}$; $в$ — $A \cap \bar{A}$; $г$ — $A \cup \bar{A}$

элементов на оси абсцисс. Такое представление делает наглядными простые логические операции над нечеткими множествами (см. рис. 1.3).

На рис. 1.3а заштрихованная часть соответствует нечеткому множеству A и, если говорить точно, изображает область значений A и всех нечетких множеств, содержащихся в A . На рис. 1.3б, в, г даны $\bar{A}$, $A \cap \bar{A}$, $A \cup \bar{A}$.

Свойства операций $\cup$ и $\cap$

Пусть A, B, C — нечеткие множества, тогда выполняются следующие свойства:

- 1)
$$\left. \begin{aligned} A \cap B &= B \cap A \\ A \cup B &= B \cup A \end{aligned} \right\} \text{ коммутативность;}$$
- 2)
$$\left. \begin{aligned} (A \cap B) \cap C &= A \cap (B \cap C) \\ (A \cup B) \cup C &= A \cup (B \cup C) \end{aligned} \right\} \text{ ассоциативность;}$$

- 3) $\left. \begin{array}{l} A \cap A = A \\ A \cup A = A \end{array} \right\}$ — *идемпотентность*;
- 4) $\left. \begin{array}{l} A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \\ A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \end{array} \right\}$ — *дистрибутивность*;
- 5) $A \cup \emptyset = A$, где $\emptyset$ — *пустое множество*, т.е. $\mu_{\emptyset}(x) = 0$ $\forall x \in E$;
- 6) $A \cap \emptyset = \emptyset$;
- 7) $A \cap E = A$, где E — *универсальное множество*;
- 8) $A \cup E = E$;
- 9) $\left. \begin{array}{l} \overline{A \cap B} = \overline{A} \cup \overline{B} \\ \overline{A \cup B} = \overline{A} \cap \overline{B} \end{array} \right\}$ — *теоремы де Моргана*.

В отличие от четких множеств, для нечетких множеств в общем случае:

$$A \cap \overline{A} \neq \emptyset, \quad A \cup \overline{A} \neq E$$

(что, в частности, проиллюстрировано выше в примере наглядного представления нечетких множеств).

З а м е ч а н и е. Введенные выше операции над нечеткими множествами основаны на использовании операций $\max$ и $\min$. В теории нечетких множеств разрабатываются вопросы построения обобщенных, параметризованных операторов пересечения, объединения и дополнения, позволяющих учесть разнообразные смысловые оттенки соответствующих им связок «и», «или», «не».

Один из подходов к операторам пересечения и объединения заключается в их определении в *классе треугольных норм и ко-норм*.

Треугольной нормой (*t-нормой*) называется двуместная действительная функция $T: [0, 1] \times [0, 1] \rightarrow [0, 1]$, удовлетворяющая следующим условиям:

- 1) $T(0, 0) = 0$; $T(\mu_A, 1) = \mu_A$; $T(1, \mu_A) = \mu_A$ — *ограниченность*;
- 2) $T(\mu_A, \mu_B) \leq T(\mu_C, \mu_D)$, если $\mu_A \leq \mu_C$, $\mu_B \leq \mu_D$ — *монотонность*;
- 3) $T(\mu_A, \mu_B) = T(\mu_B, \mu_A)$ — *коммутативность*;
- 4) $T(\mu_A, T(\mu_B, \mu_C)) = T(T(\mu_A, \mu_B), \mu_C)$ — *ассоциативность*;

Примеры треугольных норм

$$\begin{aligned} & \min(\mu_A, \mu_B) \\ & \text{произведение } \mu_A \cdot \mu_B \\ & \max(0, \mu_A + \mu_B - 1). \end{aligned}$$

Треугольной конормой (t -конормой) называется двуместная действительная функция $S: [0, 1] \times [0, 1] \rightarrow [0, 1]$ со свойствами:

- 1) $S(1, 1) = 1$; $S(\mu_A, 0) = \mu_A$; $S(0, \mu_A) = \mu_A$ — ограниченность;
- 2) $S(\mu_A, \mu_B) \geq S(\mu_C, \mu_D)$, если $\mu_A \geq \mu_C$, $\mu_B \geq \mu_D$ — монотонность;
- 3) $S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$ — коммутативность;
- 4) $S(\mu_A, S(\mu_B, \mu_C)) = S(S(\mu_A, \mu_B), \mu_C)$ — ассоциативность.

Примеры t -конорм

$$\begin{aligned} & \max(\mu_A, \mu_B) \\ & \mu_A + \mu_B - \mu_A \cdot \mu_B \\ & \min(1, \mu_A + \mu_B). \end{aligned}$$

1.2.2. Алгебраические операции над нечеткими множествами

Алгебраическое произведение A и B обозначается $A \cdot B$ и определяется так: $\forall x \in E \mu_{A \cdot B}(x) = \mu_A(x) \mu_B(x)$.

Алгебраическая сумма этих множеств обозначается $A \hat{+} B$ и определяется так: $\forall x \in E \mu_{A \hat{+} B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \mu_B(x)$.

Для операций $\{\cdot, \hat{+}\}$ выполняются свойства:

- 1) $\left. \begin{aligned} A \cdot B &= B \cdot A \\ A \hat{+} B &= B \hat{+} A \end{aligned} \right\} \text{ коммутативность;}$
- 2) $\left. \begin{aligned} (A \cdot B) \cdot C &= A \cdot (B \cdot C) \\ (A \hat{+} B) \hat{+} C &= A \hat{+} (B \hat{+} C) \end{aligned} \right\} \text{ ассоциативность;}$
- 3) $A \cdot \emptyset = \emptyset$, $A \hat{+} \emptyset = A$, $A \cdot E = A$, $A \hat{+} E = E$;
- 4) $\left. \begin{aligned} \overline{A \cdot B} &= \overline{A} \hat{+} \overline{B} \\ \overline{A \hat{+} B} &= \overline{A} \cdot \overline{B} \end{aligned} \right\} \text{ теоремы де Моргана.}$

Не выполняются:

- 1) $\left. \begin{aligned} A \cdot A &= A \\ A \hat{+} A &= A \end{aligned} \right\} \text{ — идемпотентность;}$
- 2) $\left. \begin{aligned} A \cdot (B \hat{+} C) &= (A \cdot B) \hat{+} (A \cdot C) \\ A \hat{+} (B \cdot C) &= (A \hat{+} B) \cdot (A \hat{+} C) \end{aligned} \right\} \text{ дистрибутивность;}$
- 3) а также $A \cdot \overline{A} = \emptyset$, $A \hat{+} \overline{A} = E$.

З а м е ч а н и е. При совместном использовании операций $\{\cup, \cap, \hat{+}, \cdot\}$ выполняются свойства:

- 1) $A \cdot (B \cup C) = (A \cdot B) \cup (A \cdot C)$;
- 2) $A \cdot (B \cap C) = (A \cdot B) \cap (A \cdot C)$;

$$3) A \hat{+} (B \cup C) = (A \hat{+} B) \cup (A \hat{+} C);$$

$$4) A \hat{+} (B \cap C) = (A \hat{+} B) \cap (A \hat{+} C).$$

На основе операции алгебраического произведения определяется операция *возведения в степень* α нечеткого множества A , где α — положительное число. Нечеткое множество A^α определяется функцией принадлежности $\mu_{A^\alpha} = \mu_A^\alpha(x)$. Частным случаем возведения в степень являются:

1) $\text{CON}(A) = A^2$ — операция *концентрирования* (уплотнения);

2) $\text{DIL}(A) = A^{0.5}$ — операция *растяжения*,
которые используются при работе с лингвистическими неопределенностями (рис. 1.4).

Умножение на число. Если α — положительное число, такое, что $\alpha \max_{x \in A} \mu_A(x) \leq 1$, то нечеткое множество αA имеет функцию принадлежности:

$$\mu_{\alpha A}(x) = \alpha \mu_A(x).$$

Выпуклая комбинация нечетких множеств. Пусть $A_1, A_2, \dots, A_n$ — нечеткие множества универсального множества E , а $\omega_1, \omega_2, \dots, \omega_n$ — неотрицательные числа, сумма которых равна 1.

Выпуклой комбинацией $A_1, A_2, \dots, A_n$ называется нечеткое множество A с функцией принадлежности:

$$\forall x \in E \quad \mu_A(x_1, x_2, \dots, x_n) = \omega_1 \mu_{A1}(x) + \omega_2 \mu_{A2}(x) + \dots + \omega_n \mu_{Ai}(x).$$

Декартово (прямое) произведение нечетких множеств. Пусть $A_1, A_2, \dots, A_n$ — нечеткие подмножества универсальных множеств $E_1, E_2, \dots, E_n$ соответственно. Декартово, или прямое

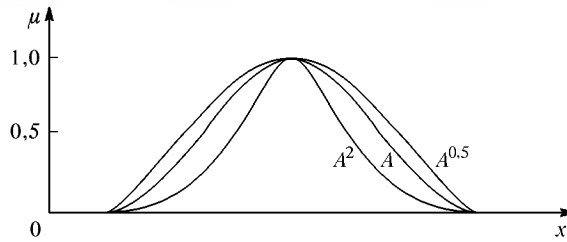


Рис. 1.4. Иллюстрация к понятию операций концентрирования (уплотнения) и растяжения

произведение $A = A_1 \times A_2 \times \dots \times A_n$ является нечетким подмножеством множества $E = E_1 \times E_2 \times \dots \times E_n$ с функцией принадлежности:

$$\mu_A(x_1, x_2, \dots, x_n) = \min(\mu_{A1}(x_1), \mu_{A2}(x_2), \dots, \mu_{Ai}(x_n)).$$

Оператор увеличения нечеткости используется для преобразования четких множеств в нечеткие и для увеличения нечеткости нечеткого множества.

Пусть A — нечеткое множество, E — универсальное множество и для всех $x \in E$ определены нечеткие множества $K(x)$. Совокупность всех $K(x)$ называется ядром оператора увеличения нечеткости Φ . Результатом действия оператора Φ на нечеткое множество A является нечеткое множество вида

$$\Phi(A, K) = \bigcup_{x \in E} \mu_A(x) K(x),$$

где $\mu_A(x)K(x)$ — произведение числа на нечеткое множество.

Пример. Пусть

$$E = \{1, 2, 3, 4\}; \quad A = 0,8/1 + 0,6/2 + 0/3 + 0/4; \quad K(1) = 1/1 + 0,4/2; \\ K(2) = 1/2 + 0,4/1 + 0,4/3; \quad K(3) = 1/3 + 0,5/4; \quad K(4) = 1/4.$$

Тогда

$$\Phi(A, K) = \mu_A(1)K(1) \cup \mu_A(2)K(2) \cup \mu_A(3)K(3) \cup \mu_A(4)K(4) = \\ = 0,8(1/1 + 0,4/2) \cup 0,6(1/2 + 0,4/1 + 0,4/3) = 0,8/1 + 0,6/2 + 0,24/3.$$

Четкое множество α -уровня (или уровня α). Множеством α -уровня нечеткого множества A универсального множества E называется *четкое* подмножество A_α универсального множества E , определяемое в виде

$$A_\alpha = \{x / \mu_A(x) \geq \alpha\},$$

где $\alpha \leq 1$.

Пример. Пусть $A = 0,2/x_1 + 0/x_2 + 0,5/x_3 + 1/x_4$, тогда $A_{0,3} = \{x_3, x_4\}$, $A_{0,7} = \{x_4\}$.

Достаточно очевидное свойство: если $\alpha_1 \geq \alpha_2$, то $A_{\alpha_1} \leq A_{\alpha_2}$.

1.3. Нечеткая и лингвистическая переменные

Понятие нечеткой и лингвистической переменных используется при описании объектов и явлений с помощью нечетких множеств.

Нечеткая переменная характеризуется тройкой $\langle \alpha, X, A \rangle$, где α — наименование переменной;

X — универсальное множество (область определения α);

A — нечеткое множество на X , описывающее ограничения (т.е. $\mu_A(x)$) на значения нечеткой переменной α .

Лингвистической переменной (ЛП) называется набор $\langle \beta, T, X, G, M \rangle$, где

β — наименование лингвистической переменной;

T — множество ее значений (терм-множество), представляющих собой наименования нечетких переменных, областью определения каждой из которых является множество X . Множество T называется базовым *терм-множеством* лингвистической переменной;

G — синтаксическая процедура, позволяющая оперировать элементами терм-множества T , в частности, генерировать новые термы (значения). Множество $T \cup G(T)$, где $G(T)$ — множество сгенерированных термов, называется расширенным терм-множеством лингвистической переменной;

M — семантическая процедура, позволяющая превратить каждое новое значение лингвистической переменной, образуемое процедурой G , в нечеткую переменную, т.е. сформировать соответствующее нечеткое множество.

З а м е ч а н и е. Чтобы избежать большого количества символов:

1) символ β используют как для названия самой переменной, так и для всех ее значений;

2) пользуются одним и тем же символом для обозначения нечеткого множества и его названия, например терм «Молодой», являющийся значением лингвистической переменной $\beta = \text{«возраст»}$, одновременно есть и нечеткое множество M («Молодой»).

Присвоение нескольких значений символам предполагает, что контекст позволяет разрешить возможные неопределенности.

П р и м е р. Пусть эксперт определяет толщину выпускаемого изделия с помощью понятий «Малая толщина», «Средняя толщина» и «Большая толщина», при этом минимальная толщина равна 10 мм, а максимальная — 80 мм.

Формализация такого описания может быть проведена с помощью следующей лингвистической переменной $\langle \beta, T, X, G, M \rangle$, где

β — толщина изделия;

$T = \{\text{«Малая толщина»}, \text{«Средняя толщина»}, \text{«Большая толщина»}\};$

$X = [10, 80];$

G — процедура образования новых термов с помощью связок «и», «или» и модификаторов типа «очень», «не», «слегка» и т.п. Например: «Малая или средняя толщина», «Очень малая толщина» и т.д.;

M — процедура задания на $X = [10, 80]$ нечетких подмножеств $A_1 = \text{«Малая толщина»}$, $A_2 = \text{«Средняя толщина»}$, $A_3 = \text{«Большая толщина»}$, а также нечетких множеств для термов из $G(T)$ в соответствии с правилами трансляции нечетких связок и модификаторов «и», «или», «не»,

«очень», «слегка» и других операций над нечеткими множествами вида: $A \cap B$, $A \cup B$, $\overline{A}$, $\text{CON } A = A^2$, $\text{DIL } A = A^{0,5}$ и т. п.

Замечание. Наряду с рассмотренными выше базовыми значениями лингвистической переменной «Толщина» ($T = \{\text{«Малая толщина»}, \text{«Средняя толщина»}, \text{«Большая толщина»}\}$) возможны значения, зависящие от области определения X . В данном случае значения лингвистической переменной «Толщина изделия» могут быть определены как «около 20 мм», «около 50 мм», «около 70 мм», т. е. в виде нечетких чисел.

Терм-множество и расширенное терм-множество в условиях примера можно характеризовать функциями принадлежности, приведенными на рис. 1.5 и 1.6.

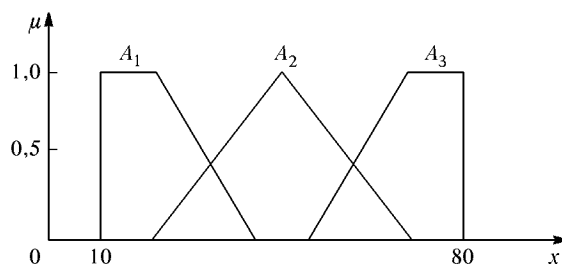


Рис. 1.5. Функции принадлежности нечетких множеств: «Малая толщина» = A_1 , «Средняя толщина» = A_2 , «Большая толщина» = A_3

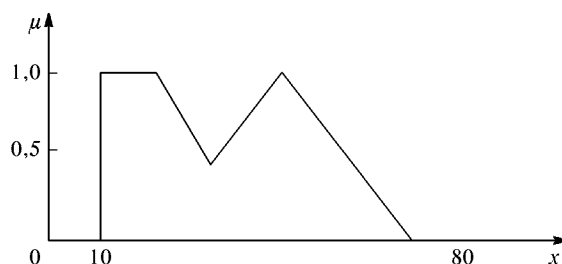


Рис. 1.6. Функция принадлежности нечеткого множества «Малая или средняя толщина» = $A_1 \cup A_2$

1.3.1. Нечеткие числа

Нечеткие числа — нечеткие переменные, определенные на числовой оси, т. е. нечеткое число определяется как нечеткое множество A на множестве действительных чисел $\mathbb{R}$ с функцией принадлежности $\mu_A(x) \in [0, 1]$, где x — действительное число, т. е. $x \in \mathbb{R}$.

Нечеткое число A *нормально*, если $\max_x \mu_A(x) = 1$; *выпуклое*, если для любых $x \leq y \leq z$ выполняется

$$\mu_A(x) \geq \mu_A(y) \wedge \mu_A(z).$$

Множество α -уровня нечеткого числа A определяется как

$$A_\alpha = \{x / \mu_A(x) \geq \alpha\}.$$

Подмножество $S_A \subset \mathbb{R}$ называется носителем нечеткого числа A , если

$$S_A = \{x / \mu_A(x) > 0\}.$$

Нечеткое число A *унимодально*, если условие $\mu_A(x) = 1$ справедливо только для одной точки действительной оси.

Выпуклое нечеткое число A называется *нечетким нулем*, если

$$\mu_A(0) = \sup_x (\mu_A(x)).$$

Нечеткое число A *положительно*, если $\forall x \in S_A, x > 0$ и *отрицательно*, если $\forall x \in S_A, x < 0$.

1.3.2. Операции над нечеткими числами. Расширенные бинарные арифметические операции (сложение, умножение и пр.) для нечетких чисел определяются через соответствующие операции для четких чисел с использованием принципа обобщения следующим образом.

Пусть A и B — нечеткие числа, и $\tilde{*}$ — нечеткая операция, соответствующая произвольной алгебраической операции $*$ над обычными числами. Тогда (используя здесь и в дальнейшем обозначения $\bigvee_x$ вместо $\max_x$ и $\bigwedge_x$ вместо $\min_x$) можно записать

$$C = A \tilde{*} B - \mu_C(z) = \bigvee_{Z=X \star Y} (\mu_A(x) \wedge \mu_B(y)).$$

Отсюда

$$C = A \tilde{+} B - \mu_C(z) = \bigvee_{Z=X+Y} (\mu_A(x) \wedge \mu_B(y)),$$

$$C = A \tilde{-} B - \mu_C(z) = \bigvee_{Z=X-Y} (\mu_A(x) \wedge \mu_B(y)),$$

$$C = A \tilde{\cdot} B - \mu_C(z) = \bigvee_{Z=X \cdot Y} (\mu_A(x) \wedge \mu_B(y)),$$

$$C = A \tilde{\div} B - \mu_C(z) = \bigvee_{Z=X \div Y} (\mu_A(x) \wedge \mu_B(y)),$$

$$C = \max(A, B) - \mu_C(z) = \bigvee_{Z=\max(X,Y)} (\mu_A(x) \wedge \mu_B(y)),$$

$$C = \min(A, B) - \mu_C(z) = \bigvee_{Z=\min(X,Y)} (\mu_A(x) \wedge \mu_B(y)).$$

1.3.3. Нечеткие числа (L–R)-типа. Нечеткие числа (L R)-типа — это разновидность нечетких чисел специального вида, т. е. задаваемых по определенным правилам с целью снижения объема вычислений при операциях над ними.

Функции принадлежности нечетких чисел (L R)-типа задаются с помощью невозрастающих на множестве неотрицательных действительных чисел функций действительного переменного $L(x)$ и $R(x)$, удовлетворяющих свойствам:

- а) $L(-x) = L(x)$, $R(-x) = R(x)$;
- б) $L(0) = R(0)$.

Очевидно, что к классу (L R)-функций относятся функции, графики которых имеют вид, приведенный на рис. 1.7.

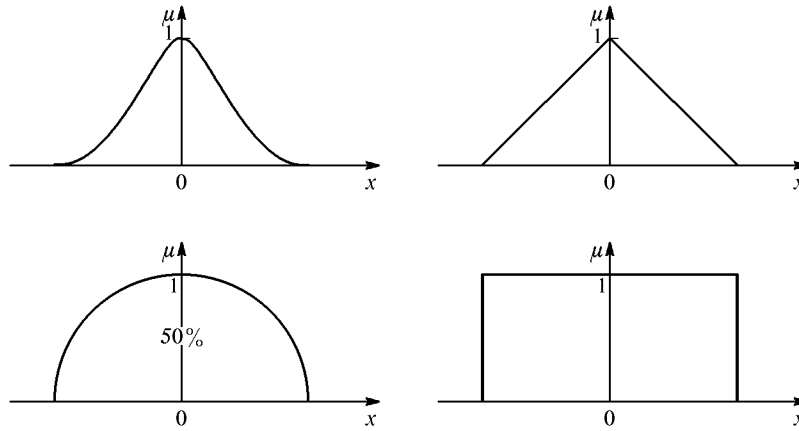


Рис. 1.7. Возможный вид (L R)-функций

Примерами аналитического задания (L R)-функций могут быть

$$L(x) = e^{-|x|^p}, \quad p \geq 0; \quad R(x) = \frac{1}{1 + |x|^p}, \quad p \geq 0,$$

и т. д.

Пусть $L(y)$ и $R(y)$ — функции (L–R)-типа (конкретные). Уни-
модальное нечеткое число A с модой a (т. е. $\mu_A(a) = 1$) с помощью

$L(y)$ и $R(y)$ задается следующим образом:

$$\mu_A(x) = \begin{cases} L\left(\frac{a-x}{\alpha}\right) & \text{при } x \leq a, \\ R\left(\frac{x-a}{\beta}\right) & \text{при } x > a, \end{cases}$$

где a — мода; $\alpha > 0$, $\beta > 0$ — левый и правый коэффициенты нечеткости.

Таким образом, при заданных $L(y)$ и $R(y)$ нечеткое число (уни-модальное) задается тройкой $A = (a, \alpha, \beta)$.

Толерантное нечеткое число задается, соответственно, четверкой параметров $A = (a_1, a_2, \alpha, \beta)$, где a_1 и a_2 — границы толерантности, т.е. в промежутке $[a_1, a_2]$ значение функции принадлежности равно 1.

Примеры графиков функций принадлежности нечетких чисел (L R)-типа приведены на рис. 1.8.

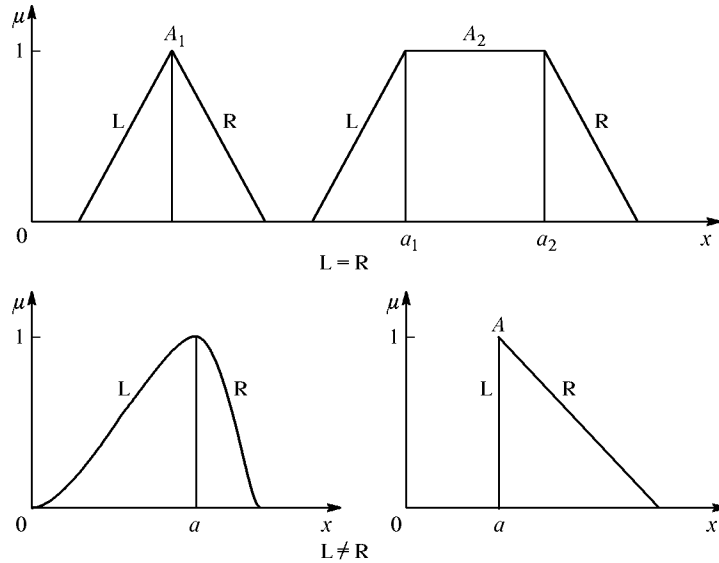


Рис. 1.8. Примеры графиков функций принадлежности нечетких чисел (L R)-типа

Отметим, что в конкретных ситуациях функции $L(y)$, $R(y)$, а также параметры α , β нечетких чисел (a, α, β) и $(a_1, a_2, \alpha, \beta)$

должны подбираться таким образом, чтобы результат операции (сложения, вычитания, деления и т. д.) был точно или приблизительно равен нечеткому числу с теми же $L(y)$ и $R(y)$, а параметры α' и β' результата не выходили за рамки ограничений на эти параметры для исходных нечетких чисел, особенно если результат в дальнейшем будет участвовать в операциях.

Замечание. Решение задач математического моделирования сложных систем с применением аппарата нечетких множеств требует выполнения большого объема операций над разного рода лингвистическими и другими нечеткими переменными. Для удобства исполнения операций, а также для ввода-вывода и хранения данных, желательно работать с функциями принадлежности стандартного вида.

Нечеткие множества, которыми приходится оперировать в большинстве задач, являются, как правило, унимодальными и нормальными. Одним из возможных методов аппроксимации унимодальных нечетких множеств является аппроксимация с помощью функций (L–R)-типа.

Примеры (L–R)-представлений некоторых лингвистических переменных приведены в табл. 1.2.

Таблица 1.2. **Возможное (L–R)-представление некоторых лингвистических переменных**

Терм ЛП	(L–R)-представление	Графическое представление
Средний	$A = (a, \alpha, \beta)_{LR}$ $\alpha = \beta > 0$	$\alpha \quad \beta$
Малый	$A = (a, \infty, \beta)_{LR}$ $\alpha = \infty$	$\alpha = \infty \quad \beta$
Большой	$A = (a, \alpha, \infty)_{LR}$ $\beta = \infty$	$\alpha \quad \beta = \infty$
Приблизительно в диапазоне	$A = (a_1, a_2, \alpha, \beta)_{LR}$ $\alpha = \beta > 0$	$\alpha \quad \beta$ $a_1 \quad a_2$
Определенный	$A = (a, 0, 0)_{LR}$ $\alpha = \beta = 0$	$\alpha = 0 \quad \beta = 0$
Разнообразный: зона полной неопределенности	$A = (a, \infty, \infty)_{LR}$ $\alpha = \beta = \infty$	$\alpha = \beta = \infty$

1.4. Нечеткие отношения

Пусть $E = E_1 \times E_2 \times \dots \times E_n$ — прямое произведение универсальных множеств и M — некоторое множество принадлежностей (например, $M = [0, 1]$). Нечеткое n -арное отношение определяется как нечеткое подмножество R на E , принимающее свои значения в M . В случае $n = 2$ и $M = [0, 1]$ нечетким отношением R между множествами $X = E_1$ и $Y = E_2$ будет называться функция $R: (X, Y) \rightarrow [0, 1]$, которая ставит в соответствие каждой паре элементов $(x, y) \in X \times Y$ величину $\mu_R(x, y) \in [0, 1]$.

Обозначение: нечеткое отношение на $X \times Y$ запишется в виде

$$x \in X, y \in Y : xRy.$$

В случае, когда $X = Y$, т.е. X и Y совпадают, нечеткое отношение $R: X \times X \rightarrow [0, 1]$ называется нечетким отношением на множестве X .

Примеры

1) Пусть $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3, y_4\}$, $M = [0, 1]$. Нечеткое отношение $R = XRY$ может быть задано, к примеру, табл. 1.3.

Таблица 1.3. Задание нечеткого отношения

	y_1	y_2	y_3	y_4
x_1	0	0	0,1	0,3
x_2	0	0,8	1	0,7
x_3	1	0,5	0,6	1

2) Пусть $X = Y = (-\infty, \infty)$, т.е. множество всех действительных чисел. Отношение $x \gg y$ (x много больше y) можно задать функцией принадлежности:

$$\mu_R = \begin{cases} 0, & \text{если } x \leq y, \\ \frac{1}{1 + (1/(x - y)^2)}, & \text{если } y < x. \end{cases}$$

3) Отношение R , для которого $\mu_R(x, y) = e^{-k(x-y)^2}$, при достаточно больших k можно интерпретировать так: « x и y близкие друг к другу числа».

1.4.1. Операции над нечеткими отношениями

Объединение двух отношений R_1 и R_2 . Объединение двух отношений обозначается $R_1 \cup R_2$ и определяется выражением

$$\mu_{R_1 \cup R_2}(x, y) = \mu_{R_1}(x, y) \vee \mu_{R_2}(x, y).$$

Пересечение двух отношений. Пересечение двух отношений R_1 и R_2 обозначается $R_1 \cap R_2$ и определяется выражением

$$\mu_{R_1 \cap R_2}(x, y) = \mu_{R_1}(x, y) \wedge \mu_{R_2}(x, y).$$

Алгебраическое произведение двух отношений. Алгебраическое произведение двух отношений R_1 и R_2 обозначается $R_1 \cdot R_2$ и определяется выражением

$$\mu_{R_1 \cdot R_2}(x, y) = \mu_{R_1}(x, y) \cdot \mu_{R_2}(x, y).$$

Алгебраическая сумма двух отношений. Алгебраическая сумма двух отношений R_1 и R_2 обозначается $R_1 \dot{+} R_2$ и определяется выражением

$$\mu_{R_1 \dot{+} R_2}(x, y) = \mu_{R_1}(x, y) + \mu_{R_2}(x, y) - \mu_{R_1}(x, y) \cdot \mu_{R_2}(x, y).$$

Для введенных операций справедливы следующие свойства дистрибутивности:

$$R_1 \cap (R_2 \cup R_3) = (R_1 \cap R_2) \cup (R_1 \cap R_3),$$

$$R_1 \cup (R_2 \cap R_3) = (R_1 \cup R_2) \cap (R_1 \cup R_3),$$

$$R_1 \cdot (R_2 \cup R_3) = (R_1 \cdot R_2) \cup (R_1 \cdot R_3),$$

$$R_1 \cdot (R_2 \cap R_3) = (R_1 \cdot R_2) \cap (R_1 \cdot R_3),$$

$$R_1 \dot{+} (R_2 \cup R_3) = (R_1 \dot{+} R_2) \cup (R_1 \dot{+} R_3),$$

$$R_1 \dot{+} (R_2 \cap R_3) = (R_1 \dot{+} R_2) \cap (R_1 \dot{+} R_3).$$

Дополнение отношения. Дополнение отношения R обозначается $\bar{R}$ и определяется функцией принадлежности:

$$\mu_{\bar{R}}(x, y) = 1 - \mu_R(x, y).$$

Дизъюнктивная сумма двух отношений. Дизъюнктивная сумма двух отношений R_1 и R_2 обозначается $R_1 \oplus R_2$ и определяется выражением

$$R_1 \oplus R_2 = (R_1 \cap \bar{R}_2) \cup (\bar{R}_1 \cap R_2).$$

Обычное отношение, ближайшее к нечеткому. Пусть R — нечеткое отношение с функцией принадлежности $\mu_R(x, y)$. Обычное отношение, ближайшее к нечеткому, обозначается $\underline{R}$ и определяется выражением

$$\mu_{\underline{R}}(x, y) = \begin{cases} 0, & \text{если } \mu_R(x, y) < 0,5, \\ 1, & \text{если } \mu_R(x, y) > 0,5, \\ 0 \text{ или } 1, & \text{если } \mu_R(x, y) = 0,5. \end{cases}$$

По договоренности принимают $\mu_{\underline{R}}(x, y) = 0$ при $\mu_{\underline{R}}(x, y) = 0,5$.

Композиция (свертка) двух нечетких отношений. Пусть R_1 — нечеткое отношение $R_1: (X \times Y) \rightarrow [0, 1]$ между X и Y , и R_2 — нечеткое отношение $R_2: (Y \times Z) \rightarrow [0, 1]$ между Y и Z . Нечеткое отношение между X и Z , обозначаемое $R_2 \circ R_1$, определенное через R_1 и R_2 выражением

$$\mu_{R_1 \circ R_2}(x, z) = \bigvee_y (\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)),$$

называется $(\max \min)$ -композицией ($(\max \min)$ -сверткой) отношений R_1 и R_2 .

Пример. Пусть

R_1	y_1	y_2	y_3
x_1	0,1	0,7	0,4
x_2	1	0,5	0

R_2	z_1	z_2	z_3	z_4
y_1	0,9	0	1	0,2
y_2	0,3	0,6	0	0,9
y_3	0,1	1	0	0,5

Тогда

$R_1 \circ R_2$	z_1	z_2	z_3	z_4
x_1	0,3	0,6	0,1	0,7
x_2	0,9	0,5	1	0,5

При этом

$$\begin{aligned} \mu_{R_1 \circ R_2}(x_1, z_1) &= (\mu_{R_1}(x_1, y_1) \wedge \mu_{R_2}(y_1, z_1)) \vee \\ &\vee (\mu_{R_1}(x_1, y_2) \wedge \mu_{R_2}(y_2, z_1)) \vee (\mu_{R_1}(x_1, y_3) \wedge \mu_{R_2}(y_3, z_1)) = \\ &= (0,1 \wedge 0,9) \vee (0,7 \wedge 0,3) \vee (0,4 \wedge 0,1) = 0,1 \vee 0,3 \vee 0,1 = 0,3; \\ \mu_{R_1 \circ R_2}(x_1, z_3) &= 0,1; \\ &\dots\dots\dots \\ \mu_{R_1 \circ R_2}(x_2, z_5) &= 0,5. \end{aligned}$$

З а м е ч а н и е. В данном примере вначале использован «аналитический» способ композиции отношений R_1 и R_2 , т.е. i -я строка R_1 «умножается» на j -й столбец R_2 с использованием операции $\wedge$, полученный результат «свертывается» с использованием операции $\vee$ в $\mu(x_i, z_j)$.

Свойства (max–min)-композиции. Операция (max–min)-композиции ассоциативна, т. е.

$$R_3 \circ (R_2 \circ R_1) = (R_3 \circ R_2) \circ R_1,$$

дистрибутивна относительно объединения, но недистрибутивна относительно пересечения:

$$\begin{aligned} R_3 \circ (R_2 \cup R_1) &= (R_3 \circ R_2) \cup (R_3 \circ R_1), \\ R_3 \circ (R_2 \cap R_1) &\neq (R_3 \circ R_2) \cap (R_3 \circ R_1). \end{aligned}$$

Кроме того, для (max–min)-композиции выполняется следующее важное свойство: если $R_1 \subset R_2$, то $R \circ R_1 \subset R \circ R_2$.

max-композиция. В выражении

$$\mu_{R_1 \circ R_2}(x, z) = \bigvee_y (\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z))$$

для (max–min)-композиции отношений R_1 и R_2 операцию $\wedge$ можно заменить любой другой, для которой выполняются те же ограничения, что и для $\wedge$: ассоциативность и монотонность (в смысле неубывания) по каждому аргументу. Тогда

$$\mu_{R_1 \circ R_2}(x, z) = \bigvee_y (\mu_{R_1}(x, y) * \mu_{R_2}(y, z)).$$

В частности, операция $\wedge$ может быть заменена алгебраическим умножением, тогда говорят о (max–prod)-композиции.

1.5. Нечеткие выводы

Используемый в различного рода экспертных и управляющих системах механизм нечетких выводов в своей основе имеет базу знаний, формируемую специалистами предметной области в виде совокупности нечетких предикатных правил вида:

П₁: если x есть A_1 , тогда y есть B_1 ,
 П₂: если x есть A_2 , тогда y есть B_2 ,

П _{n} : если x есть A_n , тогда y есть B_n ,

где x — входная переменная (имя для известных значений данных), y — переменная вывода (имя для значения данных, которое будет вычислено); A и B — функции принадлежности, определенные соответственно на x и y .

Пример подобного правила
 Если x — низко, то y — высоко.

Приведем более детальное пояснение. Знание эксперта $A \rightarrow B$ отражает нечеткое причинное отношение предпосылки и заключения, поэтому его можно назвать нечетким отношением и обозначить через R :

$$R = A \rightarrow B,$$

где « $\rightarrow$ » называют нечеткой импликацией.

Отношение R можно рассматривать как нечеткое подмножество прямого произведения $X \times Y$ полного множества предпосылок X и заключений Y . Таким образом, процесс получения (нечеткого) результата вывода B' с использованием данного наблюдения A' и знания $A \rightarrow B$ можно представить в виде формулы

$$B' = A' \circ R = A' \circ (A \rightarrow B),$$

где « $\circ$ » — введенная выше операция свертки.

Как операцию композиции, так и операцию импликации в алгебре нечетких множеств можно реализовывать по-разному (при этом, естественно, будет разниться и итоговый получаемый результат), но в любом случае общий логический вывод осуществляется за следующие четыре этапа.

1. *Нечеткость* (введение нечеткости, фазификация, fuzzification). Функции принадлежности, определенные на входных переменных применяются к их фактическим значениям для определения степени истинности каждой предпосылки каждого правила.

2. *Логический вывод*. Вычисленное значение истинности для предпосылок каждого правила применяется к заключениям каждого правила. Это приводит к одному нечеткому подмножеству, которое будет назначено каждой переменной вывода для каждого правила. В качестве правил логического вывода обычно используются только операции $\min$ (МИНИМУМ) или prod (УМНОЖЕНИЕ). В логическом выводе МИНИМУМА функция принадлежности вывода «отсекается» по высоте, соответствующей вычисленной степени истинности предпосылки правила (нечеткая логика «И»). В логическом выводе УМНОЖЕНИЯ функция принадлежности вывода масштабируется при помощи вычисленной степени истинности предпосылки правила.

3. *Композиция*. Все нечеткие подмножества, назначенные к каждой переменной вывода (во всех правилах), объединяются вместе, чтобы формировать одно нечеткое подмножество для каждой переменной вывода. При подобном объединении обычно используются операции $\max$ (МАКСИМУМ) или sum (СУММА). При композиции МАКСИМУМА комбинированный вывод нечеткого подмножества конструируется как поточечный максимум по всем нечетким подмножествам (нечеткая логика «ИЛИ»). При композиции

СУММЫ комбинированный вывод нечеткого подмножества конструируется как поточечная сумма по всем нечетким подмножествам, назначенным переменной вывода правилами логического вывода.

4. В заключение (дополнительно) *приведение к четкости* (дефазификация, defuzzification), которое используется, когда полезно преобразовать нечеткий набор выводов в четкое число. Имеется большое количество методов приведения к четкости, некоторые из которых рассмотрены ниже.

Пример. Пусть некоторая система описывается следующими нечеткими правилами:

П₁: если x есть A , тогда w есть D ,

П₂: если y есть B , тогда w есть E ,

П₃: если z есть C , тогда w есть F ,

где x, y и z — имена входных переменных, w — имя переменной вывода, а A, B, C, D, E, F — заданные функции принадлежности (треугольной формы).

Процедура получения логического вывода иллюстрируется рис. 1.9.

Предполагается, что входные переменные приняли некоторые конкретные (четкие) значения — x_0, y_0 и z_0 .

В соответствии с приведенными этапами, на этапе 1 для данных значений и исходя из функций принадлежности A, B, C , находятся степени истинности $\alpha(x_0)$, $\alpha(y_0)$ и $\alpha(z_0)$ для предпосылок каждого из трех приведенных правил (см. рис. 1.9).

На этапе 2 происходит «отсекание» функций принадлежности заключений правил (т.е. D, E, F) на уровнях $\alpha(x_0)$, $\alpha(y_0)$ и $\alpha(z_0)$.

На этапе 3 рассматриваются усеченные на втором этапе функции принадлежности и производится их объединение с использованием операции $\max$, в результате чего получается комбинированное нечеткое подмножество, описываемое функцией принадлежности $\mu_\Sigma(w)$ и соответствующее логическому выводу для выходной переменной w .

Наконец, на 4-м этапе — при необходимости — находится четкое значение выходной переменной, например, с применением центроидного метода: четкое значение выходной переменной определяется как центр тяжести для кривой $\mu_\Sigma(w)$, т.е.

$$w_0 = \frac{\int w \mu_\Sigma(w) dw}{\int \mu_\Sigma(w) dw}.$$

Рассмотрим следующие наиболее часто используемые модификации алгоритма нечеткого вывода, полагая, для простоты, что базу знаний организуют два нечетких правила вида:

П₁: если x есть A_1 и y есть B_1 , тогда z есть C_1 ,

П₂: если x есть A_2 и y есть B_2 , тогда z есть C_2 ,

где x и y — имена входных переменных, z — имя переменной вывода, $A_1, A_2, B_1, B_2, C_1, C_2$ — некоторые заданные функции принадлежности, при этом четкое значение z_0 необходимо определить на основе приведенной информации и четких значений x_0 и y_0 .

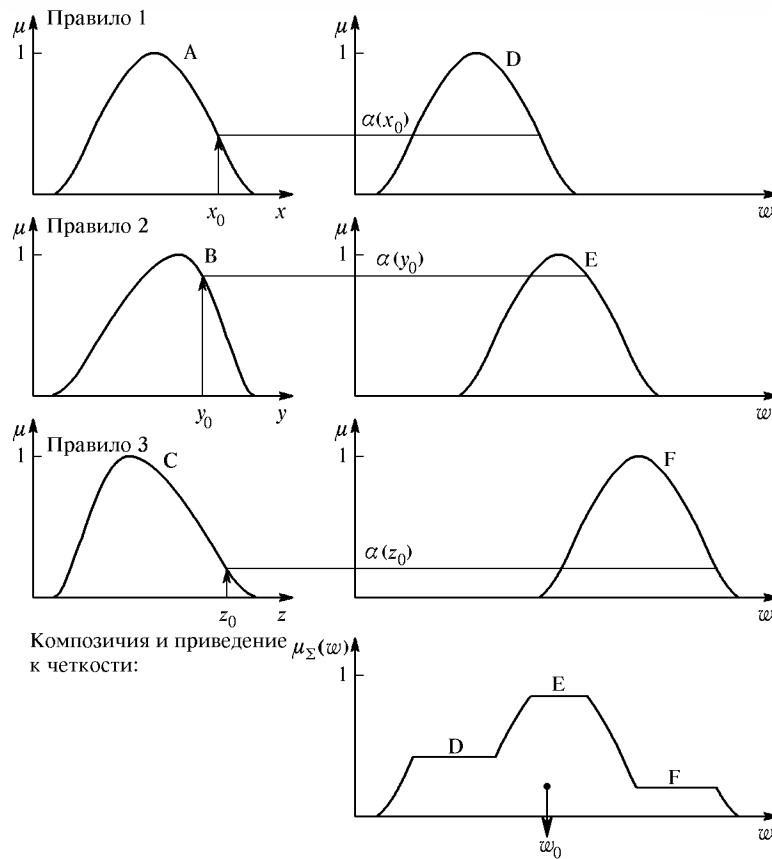


Рис. 1.9. Иллюстрация к процедуре логического вывода

1.5.1. Алгоритм Mamdani. Данный алгоритм соответствует рассмотренному примеру и рис. 1.9. В рассматриваемой ситуации он математически может быть описан следующим образом.

1. Нечеткость: находятся степени истинности для предпосылок каждого правила: $A_1(x_0), A_2(x_0), B_1(y_0), B_2(y_0)$.

2. Нечеткий вывод: находятся уровни «отсечения» для предпосылок каждого из правил (с использованием операции

МИНИМУМ)

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),$$

$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

где через « $\wedge$ » обозначена операция логического минимума ($\min$), затем находятся «усеченные» функции принадлежности

$$C'_1(z) = (\alpha_1 \wedge C_1(z)),$$

$$C'_2(z) = (\alpha_2 \wedge C_2(z)).$$

3. Композиция: с использованием операции МАКСИМУМ ($\max$, далее обозначаемой как « $\vee$ ») производится объединение найденных усеченных функций, что приводит к получению итогового нечеткого подмножества для переменной выхода с функцией принадлежности

$$\mu_{\Sigma}(z) = C(z) = C'_1(z) \vee C'_2(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)).$$

4. Наконец, приведение к четкости (для нахождения z_0) проводится, например, центроидным методом.

1.5.2. Алгоритм Tsukamoto. Исходные посылки — как у предыдущего алгоритма, но в данном случае предполагается, что функции $C_1(z)$, $C_2(z)$ являются монотонными.

1. Первый этап такой же, как в алгоритме Mamdani.

2. На втором этапе сначала находятся (как в алгоритме Mamdani) уровни «отсечения» α_1 и α_2 , а затем посредством решения уравнений

$$\alpha_1 = C_1(z_1), \quad \alpha_2 = C_2(z_2)$$

четкие значения (z_1 и z_2) для каждого из исходных правил.

3. Определяется четкое значение переменной вывода (как взвешенное среднее z_1 и z_2):

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2};$$

в общем случае (дискретный вариант центроидного метода)

$$z_0 = \frac{\sum_{i=1}^n \alpha_i z_i}{\sum_{i=1}^n \alpha_i}.$$

Пример. Пусть имеем $A_1(x_0) = 0,7$, $A_2(x_0) = 0,6$, $B_1(y_0) = 0,3$, $B_2(y_0) = 0,8$, соответствующие уровни отсечения

$$\alpha_1 = \min(A_1(x_0), B_1(y_0)) = \min(0,7; 0,3) = 0,3,$$

$$\alpha_2 = \min(A_2(x_0), B_2(y_0)) = \min(0,6; 0,8) = 0,6$$

и значения $z_1 = 8$ и $z_2 = 4$, найденные в результате решения уравнений

$$C_1(z_1) = 0,3, \quad C_2(z_2) = 0,6.$$

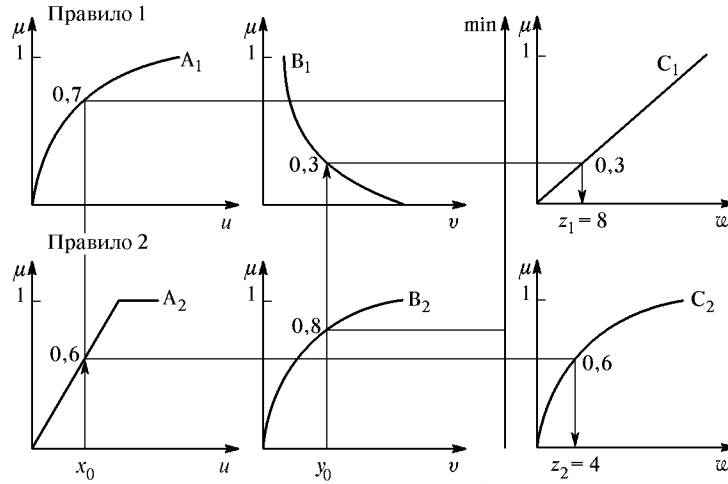


Рис. 1.10. Иллюстрации к алгоритму Tsukamoto

При этом четкое значение переменной вывода (см. рис. 1.10)

$$z_0 = (8 \cdot 0,3 + 4 \cdot 0,6) / (0,3 + 0,6) = 6.$$

1.5.3. Алгоритм Sugeno. Sugeno и Takagi использовали набор правил в следующей форме (как и раньше, приводим пример двух правил):

П₁: если x есть A_1 и y есть B_1 , тогда $z_1 = a_1x + b_1y$,

П₂: если x есть A_2 и y есть B_2 , тогда $z_2 = a_2x + b_2y$.

Представление алгоритма

1. Первый этап — как в алгоритме Mamdani.

2. На втором этапе находятся $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$ и индивидуальные выходы правил:

$$\begin{aligned} z_1^* &= a_1 x_0 + b_1 y_0, \\ z_2^* &= a_2 x_0 + b_2 y_0. \end{aligned}$$

3. На третьем этапе определяется четкое значение переменной вывода:

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2}.$$

Иллюстрирует алгоритм рис. 1.11.

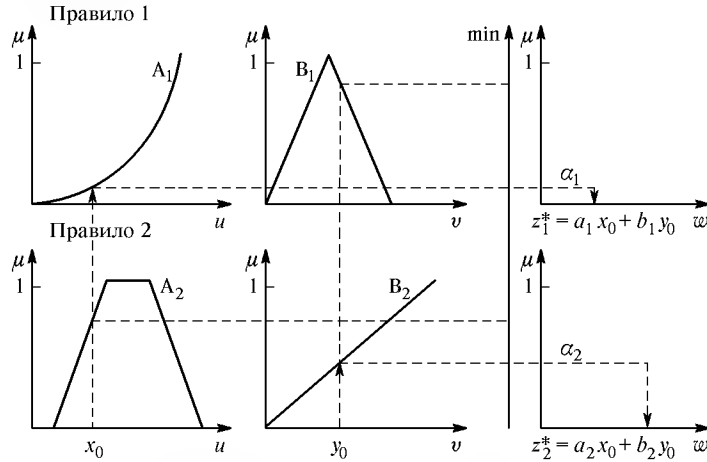


Рис. 1.11. Иллюстрация к алгоритму Sugeno

1.5.4. Алгоритм Larsen. В алгоритме Larsen нечеткая импликация моделируется с использованием оператора умножения.

Описание алгоритма

1. Первый этап — как в алгоритме Mamdani.

2. На втором этапе, как в алгоритме Mamdani вначале находятся значения

$$\begin{aligned} \alpha_1 &= A_1(x_0) \wedge B_1(y_0), \\ \alpha_2 &= A_2(x_0) \wedge B_2(y_0), \end{aligned}$$

а затем — частные нечеткие подмножества

$$\alpha_1 C_1(z), \quad \alpha_2 C_2(z).$$

3. Находится итоговое нечеткое подмножество с функцией принадлежности

$$\mu_{\Sigma}(z) = C(z) = (\alpha_1 C_1(z)) \vee (\alpha_2 C_2(z))$$

(в общем случае n правил $\mu_{\Sigma}(z) = C(z) = \bigvee_{i=1}^n (\alpha_i C_i(z))$).

4. При необходимости производится приведение к четкости (как в ранее рассмотренных алгоритмах).

Алгоритм Larsen иллюстрируется рис. 1.12.

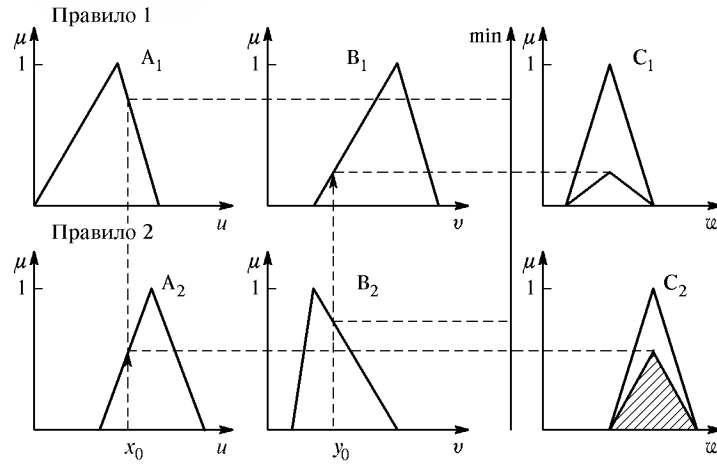


Рис. 1.12. Иллюстрация алгоритма Larsen

1.5.5. Упрощенный алгоритм нечеткого вывода. Исходные правила в данном случае задаются в виде:

П₁: если x есть A_1 и y есть B_1 , тогда $z_1 = c_1$,

П₂: если x есть A_2 и y есть B_2 , тогда $z_2 = c_2$,

где c_1 и c_2 — некоторые обычные (четкие) числа.

Описание алгоритма

1. Первый этап — как в алгоритме Mamdani.

2. На втором этапе находятся числа $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$.

3. На третьем этапе находится четкое значение выходной переменной по формуле

$$z_0 = \frac{\alpha_1 c_1 + \alpha_2 c_2}{\alpha_1 + \alpha_2},$$

или в общем случае наличия n правил по формуле

$$z_0 = \frac{\sum_{i=1}^n \alpha_i c_i}{\sum_{i=1}^n \alpha_i}.$$

Иллюстрация алгоритма приведена на рис. 1.13.

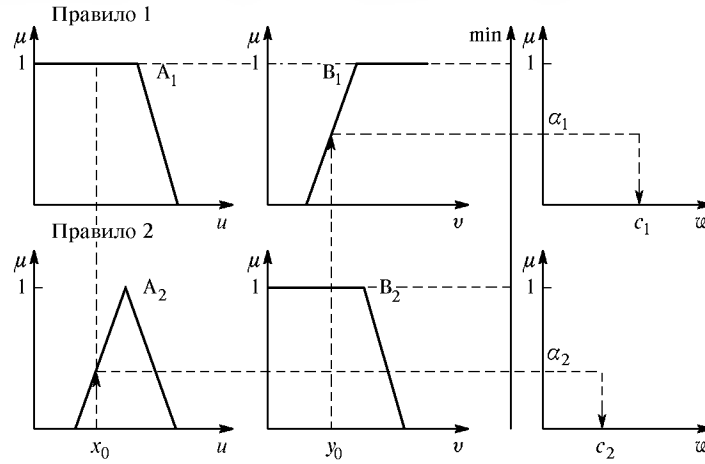


Рис. 1.13. Иллюстрация упрощенного алгоритма нечеткого вывода

1.5.6. Методы приведения к четкости

1. Выше уже был рассмотрен один из данных методов — центроидный. Приведем соответствующие формулы еще раз.

Для непрерывного варианта:

$$z_0 = \frac{\int_{\Omega} z C(z) dz}{\int_{\Omega} C(z) dz};$$

для дискретного варианта:

$$z_0 = \frac{\sum_{i=1}^n \alpha_i z_i}{\sum_{i=1}^n \alpha_i}.$$

2. Первый максимум (First-of-Maxima). Четкая величина переменной вывода находится как наименьшее значение, при котором

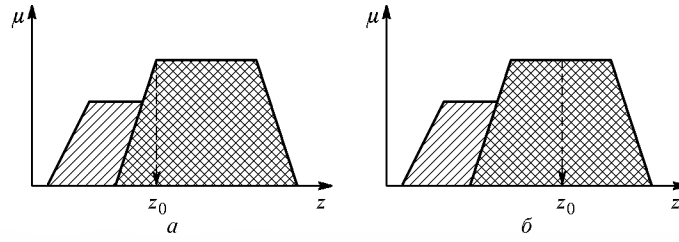


Рис. 1.14. Иллюстрация к методам приведения к четкости: *a* — первый максимум; *б* — средний максимум

достигается максимум итогового нечеткого множества, т.е. (см. рис. 1.14*a*)

$$z_0 = \min (z | C(z) = \max_u C(u)).$$

3. Средний максимум (Middle-of-Maxima). Четкое значение находится по формуле

$$z_0 = \frac{\int_G z dz}{\int_G dz},$$

где G — подмножество элементов, максимизирующих C (см. рис. 1.14*б*).

Дискретный вариант (если C — дискретно):

$$z_0 = \frac{1}{N} \sum_{j=1}^N z_j.$$

4. Критерий максимума (Max-Criterion). Четкое значение выбирается произвольно среди множества элементов, доставляющих максимум C , т.е.

$$z_0 \in \{z | C(z) = \max_u C(u)\}.$$

5. Высотная дефазификация (Height defuzzification). Элементы области определения Ω , для которых значения функции принад-

лежности меньше, чем некоторый уровень α в расчет не принимаются, и четкое значение рассчитывается по формуле

$$z_0 = \frac{\int_{C_\alpha} zC(z) dz}{\int_{C_\alpha} C(z) dz},$$

где C_α — нечеткое множество α -уровня (см. выше).

1.5.7. Нисходящие нечеткие выводы. Рассмотренные до сих пор нечеткие выводы представляют собой восходящие выводы от предпосылок к заключению. В последние годы в диагностических нечетких системах начинают применяться нисходящие выводы. Рассмотрим механизм подобного вывода на примере.

Возьмем упрощенную модель диагностики неисправности автомобиля с именами переменных:

- x_1 — неисправность аккумулятора;
- x_2 — отработка машинного масла;
- y_1 — затруднения при запуске;
- y_2 — ухудшение цвета выхлопных газов;
- y_3 — недостаток мощности.

Между x_i и y_j существуют нечеткие причинные отношения $r_{ij} = x_i \rightarrow y_j$, которые можно представить в виде некоторой матрицы $\mathbf{R}$ с элементами $r_{ij} \in [0, 1]$. Конкретные входы (предпосылки) и выходы (заключения) можно рассматривать как нечеткие множества A и B на пространствах X и Y . Отношения этих множеств можно обозначить как

$$B = A \circ \mathbf{R},$$

где, как и раньше, знак « $\circ$ » обозначает правило композиции нечетких выводов.

В данном случае направление выводов является обратным к направлению выводов для правил, т. е. в случае диагностики имеется (задана) матрица $\mathbf{R}$ (знания эксперта), наблюдаются выходы B (или симптомы) и определяются входы A (или факторы).

Пусть знания эксперта-автомеханика имеют вид

$$\mathbf{R} = \begin{bmatrix} 0,9 & 0,1 & 0,2 \\ 0,6 & 0,5 & 0,5 \end{bmatrix},$$

а в результате осмотра автомобиля его состояние можно оценить как

$$B = 0,9/y_1 + 0,1/y_2 + 0,2/y_3.$$

Требуется определить причину такого состояния:

$$A = a_1/x_1 + a_2/x_2.$$

Отношение введенных нечетких множеств можно представить в виде

$$[0,9 \ 0,1 \ 0,2] = [a_1 \ a_2] \circ \begin{bmatrix} 0,9 & 0,1 & 0,2 \\ 0,6 & 0,5 & 0,5 \end{bmatrix},$$

либо, транспонируя, в виде нечетких векторов-столбцов:

$$\begin{bmatrix} 0,9 \\ 0,1 \\ 0,2 \end{bmatrix} = \begin{bmatrix} 0,9 & 0,6 \\ 0,1 & 0,5 \\ 0,2 & 0,5 \end{bmatrix} \circ \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}.$$

При использовании (max min)-композиции последнее соотношение преобразуется к виду

$$\begin{aligned} 0,9 &= (0,9 \wedge a_1) \vee (0,6 \wedge a_2), \\ 0,1 &= (0,1 \wedge a_1) \vee (0,5 \wedge a_2), \\ 0,2 &= (0,2 \wedge a_1) \vee (0,5 \wedge a_2). \end{aligned}$$

При решении данной системы заметим прежде всего, что в первом уравнении второй член правой части не влияет на правую часть, поэтому

$$0,9 = 0,9 \wedge a_1, \quad a_1 \geq 0,9.$$

Из второго уравнения получим:

$$0,1 \geq 0,5 \wedge a_2, \quad a_2 \leq 0,1.$$

Полученное решение удовлетворяет третьему уравнению, таким образом имеем:

$$0,9 \leq a_1 \leq 1,0, \quad 0 \leq a_2 \leq 0,1,$$

т.е. лучше заменить аккумулятор (a_1 — параметр неисправности аккумулятора, a_2 — параметр отработки машинного масла).

На практике в задачах, подобных рассмотренной, количество переменных может быть существенным, могут одновременно использоваться различные композиции нечетких выводов, сама схема выводов может быть многокаскадной. Общих методов решения подобных задач в настоящее время, по-видимому, не существует.

1.6. Пример: нечеткий регулятор

Приведем еще один пример использования аппарата нечеткой логики, на этот раз в задаче управления. Рассмотрим замкнутую систему регулирования, представленную на рис. 1.15, где через O обозначен объект управления, через P — регулятор, а через u , y , e , x соответственно, входной сигнал системы, ее выходной сигнал, сигнал ошибки (рассогласования), поступающий на вход регулятора, и выходной сигнал регулятора.

В рассматриваемой системе регулятор вырабатывает управляющий сигнал x в соответствии с выбранным алгоритмом регулирования, например, пропорционально сигналу ошибки, либо ее

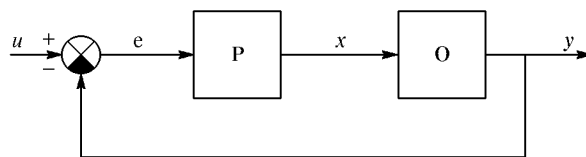


Рис. 1.15. Структура замкнутой системы управления

интегралу и т. п. Покажем, что в данном случае для выработки такого сигнала применимы рассмотренные выше методы аппарата нечеткой логики.

Предположим, что функции регулятора выполняет микроконтроллер, при этом аналоговый сигнал e ограничен диапазоном $[-1, 1]$ и преобразуется в цифровую форму аналого-цифровым преобразователем (АЦП) с дискретностью 0,25, а выходной сигнал регулятора x формируется с помощью цифроаналогового преобразователя и имеет всего 5 уровней: -1 , $-0,5$, 0 , $0,5$, 1 .

Принимая во внимание данные уровни, введем лингвистические переменные:

- A_1 : большой положительный,
- A_2 : малый положительный,
- A_3 : нулевой,
- A_4 : малый отрицательный,
- A_5 : большой отрицательный,

и на дискретном множестве возможных значений сигнала рассогласования e определим функции принадлежности так, как это приведено в табл. 1.4.

Предположим, далее, что функционирование регулятора определяется следующими правилами (надо сказать, типичными для задача управления):

- Π_1 : если $e = A_3$ и $\Delta e = A_3$, то $x = 0$,
 Π_2 : если $e = A_2$ и $\Delta e = A_2$, то $x = -0,5$,
 Π_3 : если $e = A_4$ и $\Delta e = A_4$, то $x = 1$,
 Π_4 : если $e = A_1$ и $\Delta e = A_1$, то $x = -1$,

где Δe — первая разность сигнала ошибки в текущий дискретный момент времени.

Таблица 1.4. Значения функций принадлежности

	-1	-0,75	-0,5	-0,25	0	0,25	0,5	0,75	1
$A_1(e)$	0	0	0	0	0	0	0,3	0,7	1
$A_2(e)$	0	0	0	0	0,3	0,7	1	0,7	0,3
$A_3(e)$	0	0	0,3	0,7	1	0,7	0,3	0	0
$A_4(e)$	0,3	0,7	1	0,7	0,3	0	0	0	0
$A_5(e)$	1	0,7	0,3	0	0	0	0	0	0

Заметим, что набор правил может быть, вообще говоря, и каким-то другим. Если, например, используется упрощенный алгоритм нечеткого вывода, то при значениях, скажем, $e = 0,25$ и $\Delta e = 0,5$ имеем:

$$\begin{aligned}
 \alpha_1 &= \min(0,7; 0,3) = 0,3 \quad \text{и} \quad x_1 = 0, \\
 \alpha_2 &= \min(0,7; 1) = 0,7 \quad \text{и} \quad x_2 = -0,5, \\
 \alpha_3 &= \min(0; 0) = 0 \quad \text{и} \quad x_3 = 1, \\
 \alpha_4 &= \min(0; 0,3) = 0,3 \quad \text{и} \quad x_4 = -1,
 \end{aligned}$$

и выход регулятора

$$x = \frac{0,3 \cdot 0 + 0,7 \cdot (-0,5) + 0 \cdot 1 + 0,3 \cdot (-1)}{0,3 + 0,7 + 0 + 0,3} = \frac{-0,65}{1,3} = -0,5.$$

Аналогичным образом значения выходного сигнала регулятора рассчитываются при других значениях e и Δe .

Отметим, что при проектировании подобных («нечетких») регуляторов основным (и не формализуемым) этапом является задание набора нечетких правил. Другие аспекты: выбор формы функций принадлежности, алгоритма приведения к четкости и т. п. представляются задачами более простыми.

1.7. Эффективность систем принятия решений, использующих методы нечеткой логики

Возможность использования аппарата нечеткой логики базируется на следующих результатах.

1. В 1992 г. Wang показал, что нечеткая система, использующая набор правил

П₁: если x_i есть A_i и y_i есть B_i , тогда z_i есть C_i , $i = 1, 2, \dots, n$, при:

1) гауссовских функциях принадлежности

$$A_i(x) = \exp\left(-\frac{1}{2}\left(\frac{x - \alpha_{i1}}{\beta_{i1}}\right)^2\right), \quad B_i(y) = \exp\left(-\frac{1}{2}\left(\frac{y - \alpha_{i2}}{\beta_{i2}}\right)^2\right),$$

$$C_i(z) = \exp\left(-\frac{1}{2}\left(\frac{z - \alpha_{i3}}{\beta_{i3}}\right)^2\right);$$

2) композиции в виде произведения

$$(A_i(x) \text{ and } B_i(y)) = A_i(x)B_i(y);$$

3) импликации в форме (Larsen)

$$(A_i(x) \text{ and } B_i(y)) \rightarrow C_i(z) = A_i(x)B_i(y)C_i(z);$$

4) центроидном методе приведения к четкости

$$z_0 = \frac{\sum_{i=1}^n \alpha_i A_i B_i}{\sum_{i=1}^n A_i B_i},$$

где α_i — центры C_i ; является универсальным аппроксиматором, т.е. может аппроксимировать любую непрерывную функцию на компакте U с произвольной точностью (естественно, при $n \rightarrow \infty$).

Иначе говоря, Wang доказал теорему: для каждой вещественной непрерывной функции g , заданной на компакте U , и для произвольного $\varepsilon > 0$ существует нечеткая экспертная система, формирующая выходную функцию $f(x)$ такую, что

$$\sup_{x \in U} \|g(x) - f(x)\| \leq \varepsilon,$$

где $\|\cdot\|$ — символ принятого расстояния между функциями.

2. В 1995 г. Castro показал, что логический контроллер Mamdani при:

1) симметричных треугольных функциях принадлежности:

$$A_i(x) = \begin{cases} \frac{1 - |a_i - x|}{\alpha_i}, & \text{если } |a_i - x| \leq \alpha_i, \\ 0, & \text{в противном случае,} \end{cases}$$

$$B_i(y) = \begin{cases} \frac{1 - |b_i - y|}{\beta_i}, & \text{если } |b_i - y| \leq \beta_i, \\ 0, & \text{в противном случае,} \end{cases}$$

$$C_i(z) = \begin{cases} \frac{1 - |c_i - z|}{\gamma_i}, & \text{если } |c_i - z| \leq \gamma_i, \\ 0, & \text{в противном случае;} \end{cases}$$

2) композиции с использованием операции $\min$:

$$(A_i(x) \text{ and } B_i(y)) = \min(A_i(x)B_i(y));$$

3) импликации в форме Mamdani и центроидного метода приведения к четкости:

$$z_0 = \frac{\sum_{i=1}^n c_i \min(A_i(x)B_i(y))}{\sum_{i=1}^n \min(A_i(x)B_i(y))},$$

где c_i — центры C_i ; также является универсальным аппроксиматором.

Вообще говоря, системы с нечеткой логикой целесообразно применять для сложных процессов, когда нет простой математической модели; если экспертные знания об объекте или о процессе можно сформулировать только в лингвистической форме.

Данные системы применять нецелесообразно, когда требуемый результат может быть получен каким-либо другим (стандартным) путем, или когда для объекта или процесса уже найдена адекватная и легко исследуемая математическая модель.

Отметим, что *основные недостатки* систем с нечеткой логикой связаны с тем, что:

- исходный набор постулируемых нечетких правил формулируется экспертом-человеком и может оказаться неполным или противоречивым;

- вид и параметры функций принадлежности, описывающих входные и выходные переменные системы, выбираются субъективно и могут оказаться не вполне отражающими реальную действительность.

Глава 2

ОСНОВНЫЕ ПОЛОЖЕНИЯ ТЕОРИИ НЕЙРОННЫХ СЕТЕЙ

Под нейронными сетями (НС) подразумеваются вычислительные структуры, которые моделируют простые биологические процессы, обычно ассоциируемые с процессами человеческого мозга. Адаптируемые и обучаемые, они представляют собой распараллеленные системы, способные к обучению путем анализа положительных и отрицательных воздействий. Элементарным преобразователем в данных сетях является искусственный нейрон или просто нейрон, названный так по аналогии с биологическим прототипом.

К настоящему времени предложено и изучено большое количество моделей нейроподобных элементов и нейронных сетей, ряд из которых рассмотрен в настоящей главе.

Термин «нейронные сети» сформировался в 40-х годах XX века в среде исследователей, изучавших принципы организации и функционирования биологических нейронных сетей. Основные результаты, полученные в этой области, связаны с именами американских исследователей У. Маккалоха, Д. Хебба, Ф. Розенблатта, М. Минского, Дж. Хопфилда и др.

Представим некоторые проблемы, решаемые в контексте НС и представляющие интерес для пользователей.

Классификация образов. Задача состоит в указании принадлежности входного образа (например, речевого сигнала или рукописного символа), представленного вектором признаков, одному или нескольким предварительно определенным классам. К известным приложениям относятся распознавание букв, распознавание речи, классификация сигнала электрокардиограммы, классификация клеток крови.

Кластеризация/категоризация. При решении задачи кластеризации, которая известна также как классификация образов «без учителя», отсутствует обучающая выборка с метками классов. Алгоритм кластеризации основан на подобию образов и размещает близкие образы в один кластер. Известны случаи приме-

нения кластеризации для извлечения знаний, сжатия данных и исследования свойств данных.

Аппроксимация функций. Предположим, что имеется обучающая выборка $((x_1, y_1), (x_2, y_2), \dots, (x_N, y_N))$ (пары данных вход-выход), которая генерируется неизвестной функцией $F(x)$, искаженной шумом. Задача аппроксимации состоит в нахождении оценки неизвестной функции $F(x)$. Аппроксимация функций необходима при решении многочисленных инженерных и научных задач моделирования.

Предсказание/прогноз. Пусть заданы n дискретных отсчетов $\{y(t_1), y(t_2), \dots, y(t_k)\}$ в последовательные моменты времени $t_1, t_2, \dots, t_k$. Задача состоит в предсказании значения $y(t_{k+1})$ в некоторый будущий момент времени t_{k+1} . Предсказание/прогноз имеет значительное влияние на принятие решений в бизнесе, науке и технике. Предсказание цен на фондовой бирже и прогноз погоды являются типичными приложениями техники предсказания/прогноза.

Оптимизация. Многочисленные проблемы в математике, статистике, технике, науке, медицине и экономике могут рассматриваться как проблемы оптимизации. Задачей алгоритма оптимизации является нахождение такого решения, которое удовлетворяет системе ограничений и максимизирует или минимизирует целевую функцию. Известная задача коммивояжера является классическим примером задачи оптимизации.

Память, адресуемая по содержанию. В модели вычислений фон Неймана обращение к памяти доступно только посредством адреса, который не зависит от содержания памяти. Более того, если допущена ошибка в вычислении адреса, то может быть найдена совершенно иная информация. Ассоциативная память, или память, адресуемая по содержанию, доступна по указанию заданного содержания. Содержимое памяти может быть вызвано даже по частичному входу или искаженному содержанию. Ассоциативная память чрезвычайно желательна при создании мультимедийных информационных баз данных.

Управление. Рассмотрим динамическую систему, заданную совокупностью $\{u(t), y(t)\}$, где $u(t)$ является входным управляющим воздействием, а $y(t)$ — выходом системы в момент времени t . В системах управления с эталонной моделью целью управления является расчет такого входного воздействия $u(t)$, при котором система следует по желаемой траектории, диктуемой эталонной моделью. Примером является оптимальное управление двигателем.

2.1. Биологический нейрон

Нервная система и мозг человека состоят из нейронов, соединенных между собой нервными волокнами. Нервные волокна способны передавать электрические импульсы между нейронами. Все процессы передачи раздражений от нашей кожи, ушей и глаз к мозгу, процессы мышления и управления действиями — все это реализовано в живом организме как передача электрических импульсов между нейронами.

Нейрон (нервная клетка) является особой биологической клеткой, которая обрабатывает информацию (рис. 2.1). Он состоит из тела и отростков нервных волокон двух типов — дендритов, по

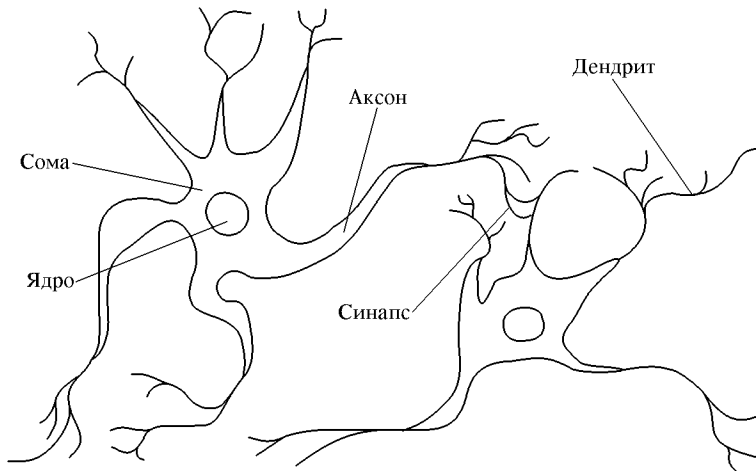


Рис. 2.1. Взаимосвязь биологических нейронов

которым принимаются импульсы, и единственного аксона, по которому нейрон может передавать импульс. Тело нейрона включает ядро, которое содержит информацию о наследственных свойствах, и плазму, обладающую молекулярными средствами для производства необходимых нейрону материалов. Нейрон получает сигналы (импульсы) от аксонов других нейронов через дендриты (приемники) и передает сигналы, сгенерированные телом клетки, вдоль своего аксона (передатчик), который в конце разветвляется на волокна. На окончаниях этих волокон находятся специальные образования — синапсы, которые влияют на силу импульса.

Синапс является элементарной структурой и функциональным узлом между двумя нейронами (волокно аксона одного нейрона и

дендрит другого). Когда импульс достигает синаптического окончания, высвобождаются определенные химические вещества, называемые нейротрансмиттерами. Нейротрансмиттеры диффундируют через синаптическую щель, возбуждая или затормаживая, в зависимости от типа синапса, способность нейрона-приемника генерировать электрические импульсы. Результативность синапса может настраиваться проходящими через него сигналами, так что синапсы могут обучаться в зависимости от активности процессов, в которых они участвуют. Эта зависимость от предыстории действует как память, которая, возможно, ответственна за память человека. Важно отметить, что веса синапсов могут изменяться со временем, что изменяет и поведение соответствующего нейрона.

2.2. Структура и свойства искусственного нейрона

Нейрон — это составная часть нейронной сети. На рис. 2.2 показана его структура.

В состав нейрона входят умножители (синапсы), сумматор и нелинейный преобразователь. Синапсы осуществляют связь между нейронами и умножают входной сигнал на число, характеризую-

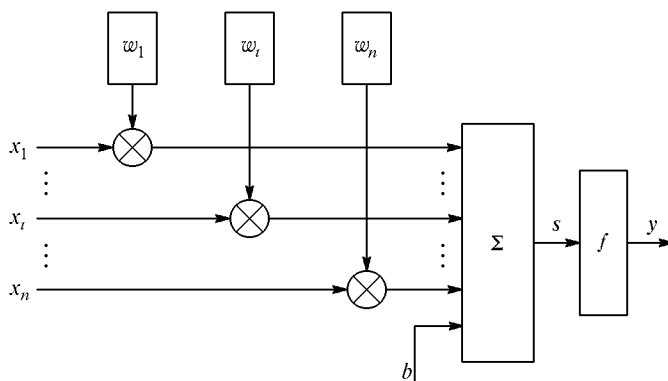


Рис. 2.2. Структура искусственного нейрона

щее силу связи, — вес синапса. Сумматор выполняет сложение сигналов, поступающих по синаптическим связям от других нейронов, и внешних входных сигналов. Нелинейный преобразователь реализует нелинейную функцию одного аргумента — выхода сумматора. Эта функция называется «функция активации» или «передаточная функция» нейрона. Нейрон в целом реализует скалярную функцию векторного аргумента. Математическая модель

нейрона описывается соотношениями

$$s = \sum_{i=1}^n w_i x_i + b,$$

где w_i — вес синапса ($i = 1, \dots, n$); b — значение смещения; s — результат суммирования; x_i — компонента входного вектора (входной сигнал) ($i = 1, \dots, n$); y — выходной сигнал нейрона; n — число входов нейрона; f — нелинейное преобразование (функция активации или передаточная функция).

Таблица 2.1. Перечень функций активации нейронов

Название	Формула	Область значений
Пороговая	$f(s) = \begin{cases} 0, & s < \theta, \\ 1, & s \geq \theta \end{cases}$	0, 1
Знаковая (сигнатурная)	$f(s) = \begin{cases} 1, & s > 0, \\ -1, & s \leq 0 \end{cases}$	-1, 1
Сигмоидальная (логистическая)	$f(s) = \frac{1}{1 + e^{-s}}$	(0, 1)
Полулинейная	$f(s) = \begin{cases} s, & s > 0, \\ 0, & s \leq 0 \end{cases}$	(0, ∞)
Линейная	$f(s) = s$	$(-\infty, \infty)$
Радиальная базисная (гауссова)	$f(s) = e^{-s^2}$	(0, 1)
Полулинейная с насыщением	$f(s) = \begin{cases} 0, & s \leq 0, \\ s, & 0 < s < 1, \\ 1, & s \geq 1 \end{cases}$	(0, 1)
Линейная с насыщением	$f(s) = \begin{cases} -1, & s \leq -1, \\ s, & -1 < s < 1, \\ 1, & s \geq 1 \end{cases}$	(-1, 1)
Гиперболический тангенс (сигмоидальная)	$f(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$	(-1, 1)
Треугольная	$f(s) = \begin{cases} 1 - s , & s \leq 1, \\ 0, & s > 1 \end{cases}$	(0, 1)

В общем случае входной сигнал, весовые коэффициенты и значения смещения могут принимать действительные значения. Выход (y) определяется видом функции активации и может быть как

действительным, так и целым. Во многих практических задачах входы, веса и смещения могут принимать лишь некоторые фиксированные значения.

Синаптические связи с положительными весами называют возбуждающими, с отрицательными весами — тормозящими.

Таким образом, нейрон полностью описывается своими весами w_i и передаточной функцией $f(s)$. Получив набор чисел (вектор) x_i в качестве входов, нейрон выдает некоторое число y на выходе.

Описанный вычислительный элемент можно считать упрощенной математической моделью биологических нейронов — клеток, из которых состоит нервная система человека и животных.

Чтобы подчеркнуть различие нейронов биологических и математических, вторые иногда называют нейроноподобными элементами или формальными нейронами.

На входной сигнал (s) нелинейный преобразователь отвечает выходным сигналом $f(s)$, который представляет собой выход ней-

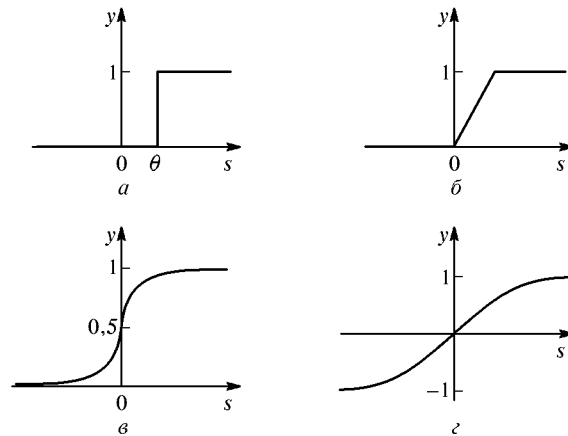


Рис. 2.3. Примеры активационных функций: a — функция единичного скачка; $б$ — линейный порог (гистерезис); $в$ — сигмоид (гиперболический тангенс); $г$ — сигмоид (логистическая)

рона y . Примеры активационных функций представлены в табл. 2.1 и на рис. 2.3.

Одной из наиболее распространенных является нелинейная функция с насыщением, так называемая логистическая функция или сигмоид (т.е. функция S-образного вида)

$$f(s) = \frac{1}{1 + e^{-as}}.$$

При уменьшении a сигмоид становится более пологим, в пределе при $a = 0$ вырождаясь в горизонтальную линию на уровне 0,5, при увеличении a сигмоид приближается по внешнему виду к функции единичного скачка с порогом θ в точке $s = 0$. Из выражения для сигмоида очевидно, что выходное значение нейрона лежит в диапазоне $[0, 1]$. Одно из ценных свойств сигмоидной функции — простое выражение для ее производной, применение которого будет рассмотрено в дальнейшем:

$$f'(s) = af(s)(1 - f(s)).$$

Следует отметить, что сигмоидная функция дифференцируема на всей оси абсцисс, что используется в некоторых алгоритмах обучения. Кроме того она обладает свойством усиливать слабые сигналы лучше, чем большие, и предотвращает насыщение от больших сигналов, так как они соответствуют областям аргументов, где сигмоид имеет пологий наклон.

Возвращаясь к общим чертам, присущим всем НС, отметим принцип параллельной обработки сигналов, который достигается путем объединения большого числа нейронов в так называемые слои и соединения определенным образом нейронов различных слоев, а также, в некоторых конфигурациях, и нейронов одного слоя между собой, причем обработка взаимодействия всех нейронов ведется послойно.

2.3. Классификация нейронных сетей и их свойства

Как отмечалось, искусственная нейронная сеть (ИНС, нейросеть) — это набор нейронов, соединенных между собой. Как правило, передаточные (активационные) функции всех нейронов в сети фиксированы, а веса являются параметрами сети и могут изменяться. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы — как внешние выходы сети. Подавая любые числа на входы сети, мы получаем какой-то набор чисел на выходах сети. Таким образом, работа нейросети состоит в преобразовании входного вектора $\mathbf{X}$ в выходной вектор $\mathbf{Y}$, причем это преобразование задается весами сети.

Практически любую задачу можно свести к задаче, решаемой нейросетью. В табл. 2.2 показано, каким образом следует сформулировать в терминах нейросети задачу распознавания рукописных букв.

Поясним, зачем требуется выбирать выход с максимальным уровнем сигнала. Дело в том, что уровень выходного сигнала, как правило, может принимать любые значения из какого-то отрезка.

Однако в данной задаче нас интересует не аналоговый ответ, а всего лишь номер категории (номер буквы в алфавите). Поэтому используется следующий подход — каждой категории сопоставляется свой выход, а ответом сети считается та категория, на чьем выходе уровень сигнала максимален. В определенном смысле уровень сигнала на выходе «А» — это достоверность того, что на вход была подана рукописная буква «А». Задачи, в которых нужно отнести входные данные к одной из известных категорий, называются задачами классификации. Изложенный подход — стандартный способ классификации с помощью нейронных сетей.

Таблица 2.2. Задача распознавания рукописных букв в терминах нейросети

Задача распознавания рукописных букв	
Дано: растровое черно-белое изображение буквы размером 30×30 пикселей	Надо: определить, какая это буква (в алфавите 33 буквы)
Формулировка для нейросети	
Дано: входной вектор из 900 двоичных символов ($900 = 30 \times 30$)	Надо: построить нейросеть с 900 входами и 33 выходами, которые помечены буквами. Если на входе сети изображение буквы «А», то максимальное значение выходного сигнала достигается на выходе «А». Аналогично сеть работает для всех 33 букв

Теперь, когда стало ясно, что именно мы хотим построить, мы можем переходить к вопросу «как строить такую сеть». Этот вопрос решается в два этапа.

1. Выбор типа (архитектуры) сети.
2. Подбор весов (обучение) сети.

На первом этапе следует выбрать следующее:

- какие нейроны мы хотим использовать (число входов, передаточные функции);
- каким образом следует соединить их между собой;
- что взять в качестве входов и выходов сети.

Эта задача на первый взгляд кажется необозримой, но, к счастью, нам необязательно придумывать нейросеть «с нуля» — существует несколько десятков различных нейросетевых архитектур, причем эффективность многих из них доказана математически. Наиболее популярные и изученные архитектуры — это многослойный персептрон, нейросеть с общей регрессией, сети Хохонена и другие, которые будут рассмотрены ниже.

На втором этапе нам следует «обучить» выбранную сеть, т.е. подобрать такие значения ее весов, чтобы сеть работала нужным образом. Необученная сеть подобна ребенку — ее можно научить чему угодно. В используемых на практике нейросетях количество весов может составлять несколько десятков тысяч, поэтому обучение — действительно сложный процесс. Для многих архитектур разработаны специальные алгоритмы обучения, которые позволяют настроить веса сети определенным образом.

В зависимости от функций, выполняемых нейронами в сети, можно выделить три их типа:

- входные нейроны — это нейроны, на которые подается входной вектор, кодирующий входное воздействие или образ внешней среды; в них обычно не осуществляется вычислительных процедур, информация передается с входа на выход нейрона путем изменения его активации;
- выходные нейроны — это нейроны, выходные значения которых представляют выход сети;
- промежуточные нейроны — это нейроны, составляющие основу искусственных нейронных сетей.

В большинстве нейронных моделей тип нейрона связан с его расположением в сети. Если нейрон имеет только выходные связи, то это входной нейрон, если наоборот — выходной нейрон. Однако может встретиться случай, когда выход топологически внутреннего нейрона рассматривается как часть выхода сети. В процессе функционирования (эволюции состояния) сети осуществляется преобразование входного вектора в выходной, т.е. некоторая переработка информации. Конкретный вид выполняемого сетью преобразования информации обуславливается не только характеристиками нейроподобных элементов (функциями активации и т.п.), но и особенностями ее архитектуры, а именно, той или иной топологией межнейронных связей, выбором определенных подмножеств нейроподобных элементов для ввода и вывода информации, способами обучения сети, наличием или отсутствием конкуренции между нейронами, направлением и способами управления и синхронизации передачи информации между нейронами.

Топология нейронных сетей. С точки зрения топологии, среди нейронных сетей, сформированных на основе нейроподобных элементов, можно выделить три основных типа:

- полносвязные сети (рис. 2.4а);
- многослойные или слоистые сети (рис. 2.4б);
- слабосвязные сети (нейронные сети с локальными связями) (рис. 2.4в).

Полносвязные сети представляют собой ИНС, каждый нейрон которой передает свой выходной сигнал остальным нейронам, в том числе и самому себе. Все входные сигналы подаются всем ней-

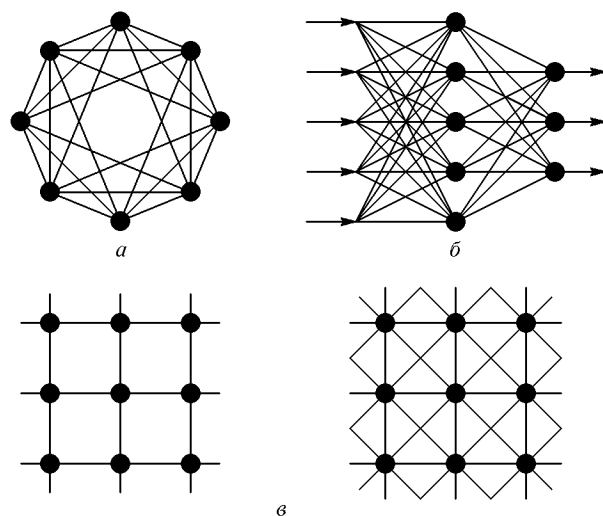


Рис. 2.4. Архитектуры нейронных сетей: а — полносвязная сеть; б — многослойная сеть с последовательными связями; в — слабосвязные сети

ронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких тактов функционирования сети.

В многослойных сетях нейроны объединяются в слои. Слой содержит совокупность нейронов с едиными входными сигналами. Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. В общем случае сеть состоит из Q слоев, пронумерованных слева направо. Внешние входные сигналы подаются на входы нейронов первого слоя (входной слой часто нумеруют как нулевой), а выходами сети являются выходные сигналы последнего слоя. Вход нейронной сети можно рассматривать как выход «нулевого слоя» вырожденных нейронов, которые служат лишь в качестве распределительных точек, суммирования и преобразования сигналов здесь не производится. Кроме входного и выходного слоев в многослойной нейронной сети есть один или несколько промежуточных (скрытых) слоев. Связи от выходов нейронов некоторого слоя q к входам нейронов следующего слоя $(q + 1)$ называются последовательными.

В свою очередь, среди слоистых сетей выделяют следующие типы.

1. *Монотонные*. Это специальный частный случай слоистых сетей с дополнительными условиями на связи и элементы. Каждый слой, кроме последнего (выходного), разбит на два блока: возбуждающий (В) и тормозящий (Т). Связи между блоками тоже разделяются на тормозящие и возбуждающие. Если от блока А к блоку С ведут только возбуждающие связи, то это означает, что любой выходной сигнал блока является монотонной неубывающей функцией любого выходного сигнала блока А. Если же эти связи только тормозящие, то любой выходной сигнал блока С является невозрастающей функцией любого выходного сигнала блока А. Для элементов монотонных сетей необходима монотонная зависимость выходного сигнала элемента от параметров входных сигналов.

2. *Сети без обратных связей*. В таких сетях нейроны входного слоя получают входные сигналы, преобразуют их и передают нейронам 1-го скрытого слоя, далее срабатывает 1-й скрытый слой и т. д. до Q -го, который выдает выходные сигналы для интерпретатора и пользователя. Если не оговорено противное, то каждый

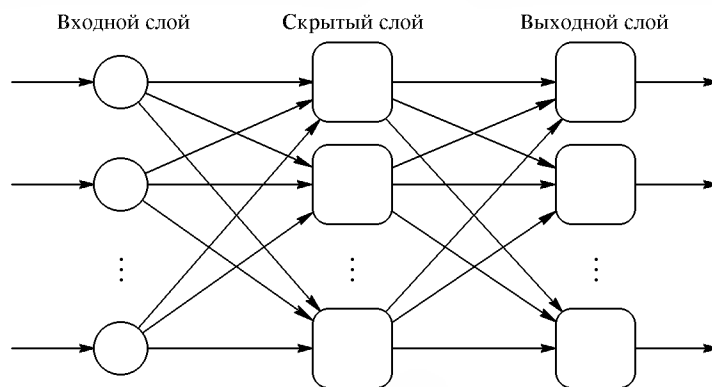


Рис. 2.5. Многослойная (двухслойная) сеть прямого распространения

выходной сигнал i -го слоя подается на вход всех нейронов $(q+l)$ -го слоя; однако возможен вариант соединения q -го слоя с произвольным $(q+p)$ -м слоем.

Следует отметить, что классическим вариантом слоистых сетей являются сети прямого распространения (рис. 2.5).

3. *Сети с обратными связями*. Это сети, у которых информация с последующих слоев передается на предыдущие.

В качестве примера сетей с обратными связями на рис. 2.6 представлены так называемые частично-рекуррентные сети Элмана и Жордана.

Известные сети можно разделить по принципу структуры нейронов в них на гомогенные или однородные и гетерогенные. Гомогенные сети состоят из нейронов одного типа с единой функцией

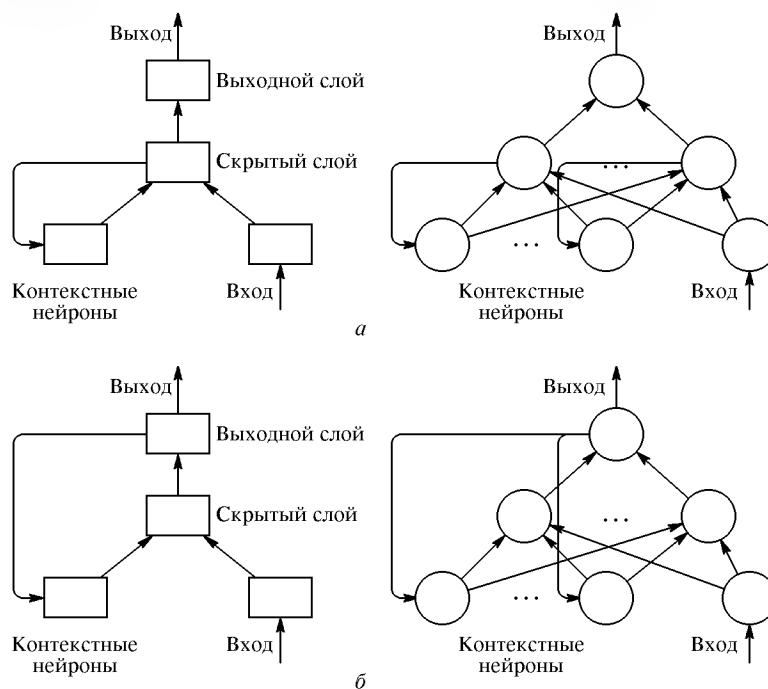


Рис. 2.6. Частично-рекуррентные сети: а — Элмана; б — Жордана

активации. В гетерогенную сеть входят нейроны с различными функциями активации.

Развивая дальше вопрос о возможной классификации НС, важно отметить существование бинарных и аналоговых сетей. Первые из них оперируют с двоичными сигналами, и выход каждого нейрона может принимать только два значения: логический ноль («заторможенное» состояние) и логическая единица («возбужденное» состояние).

Еще одна классификация делит НС на асинхронные и синхронные. В первом случае в каждый момент времени свое состояние

меняет лишь один нейрон. Во втором состоянии меняется сразу у целой группы нейронов, как правило, у всего слоя. Алгоритмически ход времени в НС задается итерационным выполнением однотипных действий над нейронами. Далее будут рассматриваться только синхронные НС.

Сети можно классифицировать также по числу слоев.

Теоретически число слоев и число нейронов в каждом слое может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированной микросхемы, на которых обычно реализуется ИНС. Чем сложнее ИНС, тем масштабнее задачи, подвластные ей.

Выбор структуры ИНС осуществляется в соответствии с особенностями и сложностью задачи. Для решения некоторых отдельных типов задач уже существуют оптимальные, на сегодняшний день, конфигурации, описанные в приложении. Если же задача не может быть сведена ни к одному из известных типов, разработчику приходится решать сложную проблему синтеза новой конфигурации. При этом он руководствуется несколькими основополагающими принципами:

- возможности сети возрастают с увеличением числа слоев сети и числа нейронов в них;
- введение обратных связей наряду с увеличением возможностей сети поднимает вопрос о так называемой динамической устойчивости сети;
- сложность алгоритмов функционирования сети (в том числе, например, введение нескольких типов синапсов возбуждающих, тормозящих и др.) также способствует усилению мощи ИНС.

Вопрос о необходимых и достаточных свойствах сети для решения того или иного рода задач представляет собой целое направление нейрокомпьютерной науки. Так как проблема синтеза ИНС сильно зависит от решаемой задачи, дать общие подробные рекомендации затруднительно. В большинстве случаев оптимальный вариант получается на основе интуитивного подбора, хотя в литературе приведены доказательства того, что для любого алгоритма существует нейронная сеть, которая может его реализовать. Остановимся на этом подробнее.

Многие задачи: распознавания образов (зрительных, речевых и т. д.), выполнения функциональных преобразований при обработке сигналов, управления, прогнозирования, идентификации сложных систем и т. д., сводятся к следующей математической постановке. Необходимо построить отображение $X \rightarrow Y$ такое, чтобы на каждый возможный входной сигнал X формировался правильный выходной сигнал Y . Отображение задается конечным

набором пар ($\langle \text{вход} \rangle$, $\langle \text{известный выход} \rangle$). Число таких пар (обучающих примеров) существенно меньше общего числа возможных сочетаний значений входных и выходных сигналов. Совокупность всех обучающих примеров носит название обучающей выборки.

В задачах распознавания образов $\mathbf{X}$ — некоторое представление образа (изображение, вектор чисел и т. д.), $\mathbf{Y}$ — номер класса, к которому принадлежит входной образ.

В задачах управления $\mathbf{X}$ — набор контролируемых параметров управляемого объекта, $\mathbf{Y}$ — код, определяющий управляющее воздействие, соответствующее текущим значениям контролируемых параметров.

В задачах прогнозирования в качестве входных сигналов используются временные ряды, представляющие значения контролируемых переменных на некотором интервале времени. Выходной сигнал — множество переменных, которое является подмножеством переменных входного сигнала.

При идентификации $\mathbf{X}$ и $\mathbf{Y}$ представляют входные и выходные сигналы системы соответственно.

Вообще говоря, большая часть прикладных задач может быть сведена к реализации некоторого сложного многомерного функционального преобразования.

В результате построения такого отображения (т. е. $\mathbf{X} \rightarrow \mathbf{Y}$) необходимо добиться того, чтобы:

- обеспечивалось формирование правильных выходных сигналов в соответствии со всеми примерами обучающей выборки;
- обеспечивалось формирование правильных выходных сигналов в соответствии со всеми возможными входными сигналами, которые не вошли в обучающую выборку.

Второе требование в значительной степени усложняет задачу формирования обучающей выборки. В общем виде эта задача в настоящее время еще не решена, однако во всех известных случаях было найдено частное решение. Дальнейшие рассуждения предполагают, что обучающая выборка уже сформирована.

Отметим, что теоретической основой для построения нейронных сетей является следующее

Утверждение. Для любого множества пар входных-выходных векторов произвольной размерности $\{(\mathbf{X}_k, \mathbf{Y}_k), k = 1 \dots N\}$ существует двухслойная однородная нейронная сеть с последовательными связями, с сигмоидальными передаточными функциями и с конечным числом нейронов, которая для каждого входного вектора $\mathbf{X}^k$ формирует соответствующий ему выходной вектор $\mathbf{Y}^k$.

Таким образом, для представления многомерных функций многих переменных может быть использована однородная нейронная сеть, имеющая всего один скрытый слой, с сигмоидальными передаточными функциями нейронов.

Для оценки числа нейронов в скрытых слоях однородных нейронных сетей можно воспользоваться формулой для оценки необходимого числа синаптических весов L_w (в многослойной сети с сигмоидальными передаточными функциями):

$$\frac{mN}{1 + \log_2 N} \leq L_w \leq m \left(\frac{N}{m} + 1 \right) (n + m + 1) + m,$$

где n — размерность входного сигнала, m — размерность выходного сигнала, N — число элементов обучающей выборки.

Оценив необходимое число весов, можно рассчитать число нейронов в скрытых слоях. Например, число нейронов в двухслойной сети составит:

$$L = \frac{L_w}{n + m}.$$

Известны и другие подобные формулы, например, вида

$$2(L + n + m) \leq N \leq 10(L + n + m),$$

$$\frac{N}{10} - n - m \leq L \leq \frac{N}{2} - n - m.$$

Точно так же можно рассчитать число нейронов в сетях с большим числом слоев, которые иногда целесообразно использовать: такие многослойные нейронные сети могут иметь меньшие размерности матриц синаптических весов нейронов одного слоя, чем двухслойные сети, реализующие то же самое отображение. К сожалению, строгая методика построения данных сетей пока отсутствует.

Отметим, что отечественному читателю приведенные результаты обычно известны в виде так называемой теоремы о полноте.

Теорема о полноте. Любая непрерывная функция на замкнутом ограниченном множестве может быть равномерно приближена функциями, вычисляемыми нейронными сетями, если функция активации нейрона дважды непрерывно дифференцируема.

Таким образом, нейронные сети являются универсальными аппроксимирующими системами.

Очевидно, что процесс функционирования ИНС, т.е. сущность действий, которые она способна выполнять, зависит от величин

синаптических связей, поэтому, задавшись определенной структурой ИНС, отвечающей какой-либо задаче, разработчик сети должен найти оптимальные значения всех переменных весовых коэффициентов (некоторые синаптические связи могут быть постоянными).

Этот этап называется обучением ИНС, и от того, насколько качественно он будет выполнен, зависит способность сети решать поставленные перед ней проблемы во время функционирования.

2.4. Обучение нейронных сетей

Обучить нейросеть значит, сообщить ей, чего мы от нее добиваемся. Этот процесс очень похож на обучение ребенка алфавиту. Показав ребенку изображение буквы «А», мы спрашиваем его: «Какая это буква?» Если ответ неверен, мы сообщаем ребенку тот ответ, который мы хотели бы от него получить: «Это буква А». Ребенок запоминает этот пример вместе с верным ответом, т. е. в его памяти происходят некоторые изменения в нужном направлении. Мы будем повторять процесс предъявления букв снова и снова до тех пор, когда все буквы будут твердо запомнены. Такой процесс называют «обучение с учителем» (рис. 2.7).

При обучении сети мы действуем совершенно аналогично. У нас имеется некоторая база данных, содержащая примеры (набор ру-

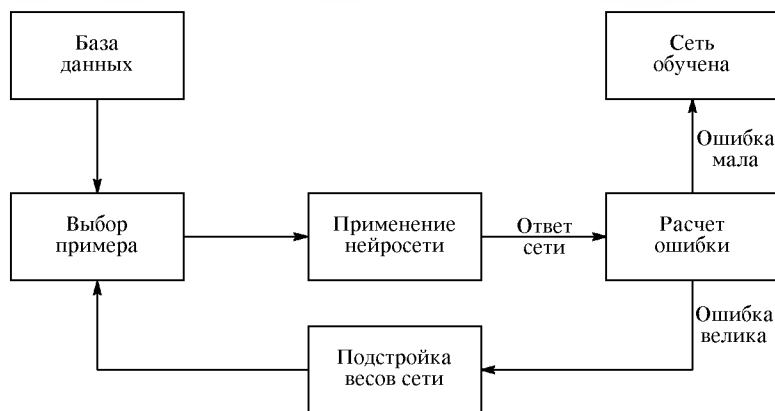


Рис. 2.7. Иллюстрация процесса обучения НС

кописных изображений букв). Предъявляя изображение буквы «А» на вход сети, мы получаем от нее некоторый ответ, не обя-

зательно верный. Нам известен и верный (желаемый) ответ в данном случае нам хотелось бы, чтобы на выходе с меткой «А» уровень сигнала был максимален. Обычно в качестве желаемого выхода в задаче классификации берут набор $(1, 0, 0, \dots)$, где 1 стоит на выходе с меткой «А», а 0 — на всех остальных выходах. Вычисляя разность между желаемым ответом и реальным ответом сети, мы получаем (для букв русского алфавита) 33 числа — вектор ошибки. Алгоритм обучения — это набор формул, который позволяет по вектору ошибки вычислить требуемые поправки для весов сети. Одну и ту же букву (а также различные изображения одной и той же буквы) мы можем предъявлять сети много раз. В этом смысле обучение скорее напоминает повторение упражнений в спорте — тренировку.

Оказывается, что после многократного предъявления примеров веса сети стабилизируются, причем сеть дает правильные ответы на все (или почти все) примеры из базы данных. В таком случае говорят, что «сеть выучила все примеры», «сеть обучена», или «сеть натренирована». В программных реализациях можно видеть, что в процессе обучения функция ошибки (например, сумма квадратов ошибок по всем выходам) постепенно уменьшается. Когда функция ошибки достигает нуля или приемлемого малого уровня, тренировку останавливают, а полученную сеть считают натренированной и готовой к применению на новых данных.

Важно отметить, что вся информация, которую сеть имеет о задаче, содержится в наборе примеров. Поэтому качество обучения сети напрямую зависит от количества примеров в обучающей выборке, а также от того, насколько полно эти примеры описывают данную задачу. Так, например, бессмысленно использовать сеть для предсказания финансового кризиса, если в обучающей выборке кризисов не представлено. Считается, что для полноценной тренировки требуется хотя бы несколько десятков (а лучше сотен) примеров.

Математически процесс обучения можно описать следующим образом.

В процессе функционирования нейронная сеть формирует выходной сигнал $\mathbf{Y}$ в соответствии с входным сигналом $\mathbf{X}$, реализуя некоторую функцию $\mathbf{Y} = G(\mathbf{X})$. Если архитектура сети задана, то вид функции G определяется значениями синаптических весов и смещений сети.

Пусть решением некоторой задачи является функция $\mathbf{Y} = F(\mathbf{X})$, заданная парами входных-выходных данных $(\mathbf{X}^1, \mathbf{Y}^1)$, $(\mathbf{X}^2, \mathbf{Y}^2)$, ..., $(\mathbf{X}^N, \mathbf{Y}^N)$, для которых $\mathbf{Y}^k = F(\mathbf{X}^k)$ ($k = 1, 2, \dots, N$).

Обучение состоит в поиске (синтезе) функции G , близкой к F в смысле некоторой функции ошибки E (см. рис. 2.7).

Если выбраны множество обучающих примеров — пар $(\mathbf{X}^k, \mathbf{Y}^k)$ (где $k = 1, 2, \dots, N$) и способ вычисления функции ошибки E , то обучение нейронной сети превращается в задачу многомерной оптимизации, имеющую очень большую размерность, при этом, поскольку функция E может иметь произвольный вид, обучение в общем случае — многоэкстремальная невыпуклая задача оптимизации.

Для решения этой задачи могут быть использованы следующие (итерационные) алгоритмы:

- алгоритмы локальной оптимизации с вычислением частных производных первого порядка;
- алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка;
- стохастические алгоритмы оптимизации;
- алгоритмы глобальной оптимизации.

К первой группе относятся: градиентный алгоритм (метод скорейшего спуска); методы с одномерной и двумерной оптимизацией целевой функции в направлении антиградиента; метод сопряженных градиентов; методы, учитывающие направление антиградиента на нескольких шагах алгоритма.

Ко второй группе относятся: метод Ньютона, методы оптимизации с разреженными матрицами Гессе, квазиньютоновские методы, метод Гаусса–Ньютона, метод Левенберга–Марквардта и др.

Стохастическими методами являются: поиск в случайном направлении, имитация отжига, метод Монте-Карло (численный метод статистических испытаний).

Задачи глобальной оптимизации решаются с помощью перебора значений переменных, от которых зависит целевая функция (функция ошибки E).

2.4.1. Алгоритм обратного распространения. Рассмотрим идею одного из самых распространенных алгоритмов обучения — алгоритм обратного распространения ошибки (back propagation). Это итеративный градиентный алгоритм обучения, который используется с целью минимизации среднеквадратичного отклонения текущего выхода и желаемого выхода многослойных нейронных сетей.

Алгоритм обратного распространения используется для обучения многослойных нейронных сетей с последовательными связями вида рис. 2.5. Как отмечено выше, нейроны в таких сетях делятся на группы с общим входным сигналом — слои, при этом на каждый нейрон первого слоя подаются все элементы внешнего

входного сигнала, а все выходы нейронов m -го слоя подаются на каждый нейрон слоя $(q + 1)$. Нейроны выполняют взвешенное (с синаптическими весами) суммирование элементов входных сигналов; к данной сумме прибавляется смещение нейрона. Над полученным результатом выполняется активационной функцией затем нелинейное преобразование. Значение функции активации есть выход нейрона.

В многослойных сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, неизвестны, и трех- или более слойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходах НС. Наиболее приемлемым вариантом обучения в таких условиях оказался градиентный метод поиска минимума функции ошибки с рассмотрением сигналов ошибки от выходов НС к ее входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Этот алгоритм обучения НС получил название процедуры обратного распространения.

В данном алгоритме функция ошибки представляет собой сумму квадратов рассогласования (ошибки) желаемого выхода сети и реального. При вычислении элементов вектора-градиента использован своеобразный вид производных функций активации сигмоидального типа. Алгоритм действует циклически (итеративно), и его циклы принято называть эпохами. На каждой эпохе на вход сети поочередно подаются все обучающие наблюдения, выходные значения сети сравниваются с целевыми значениями и вычисляется ошибка. Значение ошибки, а также градиента поверхности ошибок используется для корректировки весов, после чего все действия повторяются. Начальная конфигурация сети выбирается случайным образом, и процесс обучения прекращается, либо когда пройдено определенное количество эпох, либо когда ошибка достигнет некоторого определенного уровня малости, либо когда ошибка перестанет уменьшаться (пользователь может сам выбрать нужное условие остановки).

Приведем словесное описание алгоритма.

Шаг 1. Весам сети присваиваются небольшие начальные значения.

Шаг 2. Выбирается очередная обучающая пара $(\mathbf{X}, \mathbf{Y})$ из обучающего множества; вектор $\mathbf{X}$ подается на вход сети.

Шаг 3. Вычисляется выход сети.

Шаг 4. Вычисляется разность между требуемым (целевым, $\mathbf{Y}$) и реальным (вычисленным) выходом сети.

Шаг 5. Веса сети корректируются так, чтобы минимизировать ошибку.

драту разности желаемого выхода и выхода сети:

$$E_k = \frac{1}{2}(y^k - o^k)^2 = \frac{1}{2}(y^k - o^k(\mathbf{w}^\top \mathbf{x}^k))^2 = \frac{1}{2} \left(y^k - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}^k}} \right)^2.$$

Соответственно, суммарная функция ошибки по всем элементам выборки:

$$E = \sum_{k=1}^N E_k.$$

Очевидно, как E_k , так и E являются функциями вектора весов сети $\mathbf{w}$. Задача обучения сети сводится в данном случае к подбору такого вектора $\mathbf{w}$, при котором достигается минимум E . Данную задачу (оптимизации) будем решать градиентным методом, используя соотношение

$$\mathbf{w} := \mathbf{w} - \eta E'_k(\mathbf{w}),$$

где «:=» — оператор присвоения, $E'_k(\mathbf{w})$ — обозначение вектора-градиента, η — некоторая константа.

Представляя данный вектор в развернутом виде и учитывая приведенное выше (в § 2.2) выражение для производной сигмоидной функции, получим:

$$E'_k(\mathbf{w}) = \frac{d}{d\mathbf{w}} \left(\frac{1}{2} \left(y^k - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}^k}} \right)^2 \right) = -(y^k - o^k) o^k (1 - o^k) \mathbf{x}^k.$$

Это дает возможность записать алгоритм коррекции (подстройки) вектора весовых коэффициентов сети в форме

$$\mathbf{w} := \mathbf{w} + \eta (y^k - o^k) o^k (1 - o^k) \mathbf{x}^k = \mathbf{w} + \eta \delta_k \mathbf{x}^k,$$

где

$$\delta_k = (y^k - o^k) o^k (1 - o^k).$$

Полученные математические выражения полностью определяют алгоритм обучения рассматриваемой НС, который может быть представлен теперь в следующем виде.

1. Задаются некоторые η ($0 < \eta < 1$), $E_{\max}$ и некоторые малые случайные веса w_i сети.

2. Задаются $k = 1$ и $E = 0$.

3. Вводится очередная обучающая пара $(\mathbf{x}^k, y^k)$. Производятся обозначения

$$\mathbf{x} := \mathbf{x}^k, \quad y := y^k$$

и вычисляется величина выхода сети:

$$o = o(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}.$$

4. Обновляются (корректируются) веса:

$$\mathbf{w} := \mathbf{w} + \eta(y - o)o(1 - o)\mathbf{x}.$$

5. Корректируется (наращивается) значение функции ошибки:

$$E := E + \frac{1}{2}(y - o)^2.$$

6. Если $k < N$, тогда $k := k + 1$ и переход к шагу 3, в противном случае — переход на шаг 7.

7. Завершение цикла обучения. Если $E < E_{\max}$, то окончание всей процедуры обучения. Если $E \geq E_{\max}$, тогда начинается новый цикл обучения переходом к шагу 2.

Рассмотрим теперь более общий случай, полагая, что двухслойная НС содержит несколько (L) скрытых нейронов и один выходной (рис. 2.9).

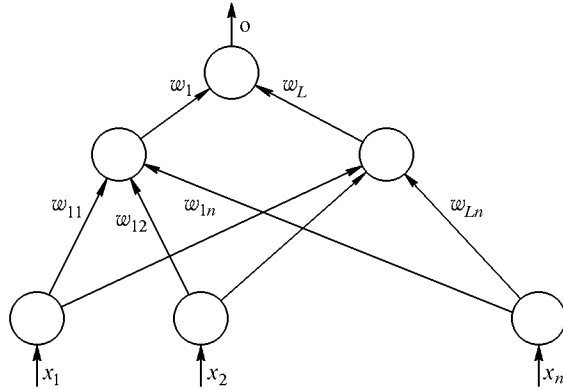


Рис. 2.9. Двухслойная нейронная сеть

В данном случае функция ошибки зависит от векторов весов скрытого слоя и вектора весов, связанных с выходным нейроном.

Выход сети описывается выражением

$$O^k = \frac{1}{1 + e^{-\mathbf{W}^\top \mathbf{o}^k}},$$

где $\mathbf{W}$ — вектор весов выходного нейрона, $\mathbf{o}^k$ — вектор выходов нейронов скрытого слоя с элементами

$$o_i^k = \frac{1}{1 + e^{-\mathbf{w}_i^\top \mathbf{x}^k}},$$

$\mathbf{w}_i$ обозначает вектор весов, связанных с i -м скрытым нейроном, $i = 1, 2, \dots, L$.

Правило корректировки весов в рассматриваемой НС также основано на минимизации квадратичной функции ошибки градиентным методом на основе выражений:

$$\begin{aligned}\mathbf{W} &:= \mathbf{W} - \eta \frac{\partial E_k(\mathbf{W}, \mathbf{w})}{\partial \mathbf{W}}, \\ \mathbf{w}_i &:= \mathbf{w}_i - \eta \frac{\partial E_k(\mathbf{W}, \mathbf{w})}{\partial \mathbf{w}_i},\end{aligned}$$

где $\eta = \text{const}$ — коэффициент скорости обучения ($0 < \eta < 1$), $i = 1, 2, \dots, L$.

Используя правило дифференцирования сложной функции и выражение для производной сигмоидной функции активации, получим:

$$\begin{aligned}\frac{\partial E_k(\mathbf{W}, \mathbf{w})}{\partial \mathbf{W}} &= \frac{1}{2} \frac{\partial}{\partial \mathbf{W}} \left(y^k - \frac{1}{1 + e^{-\mathbf{W}^\top \mathbf{o}^k}} \right)^2 = \\ &= -(y^k - O^k) O^k (1 - O^k) \mathbf{o}^k,\end{aligned}$$

откуда следует

$$\mathbf{W} := \mathbf{W} + \eta (y^k - O^k) O^k (1 - O^k) \mathbf{o}^k = \mathbf{W} + \eta \delta_k \mathbf{o}^k,$$

или в скалярной форме

$$W_i := W_i + \eta \delta_k o_i^k, \quad i = 1, 2, \dots, L,$$

где

$$\delta_k = (y^k - O^k) O^k (1 - O^k).$$

Поступая аналогично, найдем

$$\frac{\partial E_k(\mathbf{W}, \mathbf{w})}{\partial \mathbf{w}_i} = -(y^k - O^k) O^k (1 - O^k) W_i o_i^k (1 - o_i^k) \mathbf{x}^k,$$

откуда получаем

$$\mathbf{w}_i := \mathbf{w}_i + \eta \delta_k W_i o_i^k (1 - o_i^k) \mathbf{x}^k,$$

или (в скалярной форме)

$$w_{ij} := w_{ij} + \eta \delta_k W_i o_i^k (1 - o_i^k) x_j^k, \quad i = 1, 2, \dots, L, \quad j = 1, 2, \dots, n.$$

Алгоритм обучения может быть теперь представлен в виде следующих шагов.

1. Задаются некоторые η ($0 < \eta < 1$), $E_{\max}$ и некоторые малые случайные веса w_i сети.

2. Задаются $k = 1$ и $E = 0$.

3. Вводится очередная обучающая пара $(\mathbf{x}^k, y^k)$. Производятся обозначения

$$\mathbf{x} := \mathbf{x}^k, \quad y := y^k$$

и вычисляется величина выхода сети:

$$O = \frac{1}{1 + e^{-\mathbf{W}^T \mathbf{o}}},$$

где $\mathbf{W}$ — вектор весов выходного нейрона, $\mathbf{o}^k$ — вектор выходов нейронов скрытого слоя с элементами

$$o_i = \frac{1}{1 + e^{-\mathbf{w}_i^T \mathbf{x}}},$$

$\mathbf{w}_i$ обозначает вектор весов, связанных с i -м скрытым нейроном, $i = 1, 2, \dots, L$.

4. Производится корректировка весов выходного нейрона:

$$\mathbf{W} := \mathbf{W} + \eta \delta \mathbf{o},$$

где $\delta = (y - O)O(1 - O)$.

5. Корректируются веса нейронов скрытого слоя:

$$\mathbf{w}_i := \mathbf{w}_i + \eta \delta W_i o_i (1 - o_i) \mathbf{x}, \quad i = 1, 2, \dots, L.$$

6. Корректируется (наращивается) значение функции ошибки:

$$E := E + \frac{1}{2}(y - o)^2.$$

Если $k < N$, тогда $k := k + 1$ и переход к шагу 3, в противном случае переход на шаг 8.

7. Завершение цикла обучения. Если $E < E_{\max}$, то окончание всей процедуры обучения. Если $E \geq E_{\max}$, тогда начинается новый цикл обучения переходом к шагу 2.

Рассмотренная процедура может быть легко обобщена на случай сети с произвольным количеством слоев и нейронов в каждом

слое. Обратим внимание, что в данной процедуре сначала происходит коррекция весов для выходного нейрона, а затем для нейронов скрытого слоя, т.е. от конца сети к ее началу. Отсюда и название обратное распространение ошибки. Ввиду использования для обозначений греческой буквы δ , эту процедуру обучения называют еще иногда *обобщенным дельта-правилом*.

Дадим изложенному геометрическую интерпретацию.

В алгоритме *обратного распространения* вычисляется вектор градиента поверхности ошибок. Этот вектор указывает направление кратчайшего спуска по поверхности из данной точки, поэтому, если мы «немного» продвинемся по нему, ошибка уменьшится. Последовательность таких шагов (замедляющаяся по мере приближения к дну) в конце концов приведет к минимуму того или иного типа. Определенную трудность здесь представляет вопрос о том, какую нужно брать длину шагов (что определяется величиной коэффициента скорости обучения η).

При большой длине шага сходимость будет более быстрой, но имеется опасность перепрыгнуть через решение или (если поверхность ошибок имеет особо вычурную форму) уйти в неправильном направлении. Классическим примером такого явления при обучении нейронной сети является ситуация, когда алгоритм очень медленно продвигается по узкому оврагу с крутыми склонами, прыгая с одной его стороны на другую. Напротив, при маленьком шаге, вероятно, будет схвачено верное направление, однако при этом потребуются очень много итераций. На практике величина шага берется пропорциональной крутизне склона (так что алгоритм замедляет ход вблизи минимума) с некоторой константой (η), которая, как отмечалось, называется коэффициентом скорости обучения. Правильный выбор скорости обучения зависит от конкретной задачи и обычно осуществляется опытным путем; эта константа может также зависеть от времени, уменьшаясь по мере продвижения алгоритма.

Обычно этот алгоритм видоизменяется таким образом, чтобы включать слагаемое импульса (или инерции). Этот член способствует продвижению в фиксированном направлении, поэтому если было сделано несколько шагов в одном и том же направлении, то алгоритм «увеличивает скорость», что (иногда) позволяет избежать локального минимума, а также быстрее проходить плоские участки.

Таким образом, алгоритм действует итеративно, и его шаги принято называть эпохами. На каждой эпохе на вход сети поочередно подаются все обучающие наблюдения, выходные значения сети сравниваются с целевыми значениями и вычисляется

ошибка. Значение ошибки, а также градиента поверхности ошибок используется для корректировки весов, после чего все действия повторяются. Начальная конфигурация сети выбирается случайным образом, и процесс обучения прекращается, либо когда пройдено определенное количество эпох, либо когда ошибка достигнет некоторого определенного уровня малости, либо когда ошибка перестанет уменьшаться (пользователь может сам выбрать нужное условие остановки).

Классический метод обратного распространения относится к алгоритмам с линейной сходимостью и известными недостатками его являются: невысокая скорость сходимости (большое число требуемых итераций для достижения минимума функции ошибки), возможность сходиться не к глобальному, а к локальным решениям (локальным минимумам отмеченной функции), возможность паралича сети (при котором большинство нейронов функционирует при очень больших значениях аргумента функций активации, т. е. на ее пологом участке; поскольку ошибка пропорциональна производной, которая на данных участках мала, то процесс обучения практически замирает).

Для устранения этих недостатков были предложены многочисленные модификации алгоритма обратного распространения, которые связаны с использованием различных функций ошибки, различных процедур определения направления и величины шага и т. п.

Переобучение и обобщение. Одна из наиболее серьезных трудностей изложенного подхода заключается в том, что таким образом мы минимизируем не ту ошибку, которую на самом деле нужно минимизировать, — ошибку, которую можно ожидать от сети, когда ей будут подаваться совершенно новые наблюдения.

Иначе говоря, мы хотели бы, чтобы нейронная сеть обладала способностью *обобщать* результат на новые наблюдения. В действительности сеть обучается минимизировать ошибку на обучающем множестве, и в отсутствие идеального и бесконечно большого обучающего множества это совсем не то же самое, что минимизировать «настоящую» ошибку на поверхности ошибок в заранее неизвестной модели явления.

Сильнее всего это различие проявляется в проблеме переобучения, или слишком близкой подгонки. Это явление проще будет продемонстрировать не для нейронной сети, а на примере аппроксимации посредством полиномов, — при этом суть явления абсолютно та же.

Полином (или многочлен) — это выражение, содержащее только константы и целые степени независимой переменной. Примеры:

$$y = 2x + 3, \quad y = 3x^2 + 4x + 1.$$

Графики полиномов могут иметь различную форму, причем чем выше степень многочлена (и, тем самым, чем больше членов в него входит), тем более сложной может быть эта форма. Если у нас есть некоторые данные, мы можем поставить цель подогнать к ним полиномиальную кривую (модель) и получить таким образом объяснение для имеющейся зависимости. Наши данные могут быть зашумлены, поэтому нельзя считать, что самая лучшая модель задается кривой, которая в точности проходит через все имеющиеся точки. Полином низкого порядка может быть недостаточно гибким средством для аппроксимации данных, в то время как полином высокого порядка может оказаться чересчур гибким, и будет точно следовать данным, принимая при этом замысловатую форму, не имеющую никакого отношения к форме настоящей зависимости.

Нейронная сеть сталкивается с точно такой же трудностью. Сети с большим числом весов моделируют более сложные функции и, следовательно, склонны к переобучению. Сеть же с небольшим числом весов может оказаться недостаточно гибкой, чтобы смоделировать имеющуюся зависимость.

Как же выбрать «правильную» степень сложности для сети? Почти всегда более сложная сеть дает меньшую ошибку, но это может свидетельствовать не о хорошем качестве модели, а о переобучении.

Ответ состоит в том, чтобы использовать механизм контрольной кросс-проверки, при котором часть обучающих наблюдений резервируется и в обучении по алгоритму обратного распространения не используется. Вместо этого, по мере работы алгоритма, она используется для независимого контроля результата. В самом начале работы ошибки сети на обучающем и контрольном множествах будут одинаковыми (если они существенно отличаются, то, вероятно, разбиение всех наблюдений на два множества было неоднородно). По мере того как сеть обучается, ошибка обучения, естественно, убывает, и, пока обучение уменьшает действительную функцию ошибок, ошибка на контрольном множестве также будет убывать. Если же контрольная ошибка перестала убывать или даже стала расти, это указывает на то, что сеть начала слишком близко аппроксимировать данные и обучение следует остановить. Это явление чересчур точной аппроксимации в процессе обучения и называется переобучением. Если такое случилось, то обычно советуют уменьшить число скрытых элементов и/или слоев, ибо

сеть является слишком мощной для данной задачи. Если же сеть, наоборот, была взята недостаточно богатой для того, чтобы моделировать имеющуюся зависимость, то переобучения, скорее всего, не произойдет, и обе ошибки — обучения и проверки — не достигнут достаточного уровня малости.

Описанные проблемы с локальными минимумами и выбором размера сети приводят к тому, что при практической работе с нейронными сетями, как правило, приходится экспериментировать с большим числом различных сетей, порой обучая каждую из них по несколько раз (чтобы не быть введенным в заблуждение локальными минимумами) и сравнивая полученные результаты. Главным показателем качества результата является здесь контрольная ошибка. При этом в соответствии с общенаучным принципом, согласно которому при прочих равных следует предпочесть более простую модель, из двух сетей с приблизительно равными ошибками контроля имеет смысл выбрать ту, которая меньше.

Классический метод обратного распространения относится к алгоритмам с линейной сходимостью. Для увеличения скорости сходимости необходимо использовать матрицы вторых производных функции ошибки.

Были предложены многочисленные модификации алгоритма обратного распространения, которые связаны с использованием различных функций ошибки, различных процедур определения направления и величины шага.

1. Функции ошибки:

- интегральные функции ошибки по всей совокупности обучающих примеров;
- функции ошибки целых и дробных степеней.

2. Процедуры определения величины шага на каждой итерации:

- дихотомия;
- инерционные соотношения;
- отжиг.

3. Процедуры определения направления шага:

- с использованием матрицы производных второго порядка (метод Ньютона и др.);
- с использованием направлений на нескольких шагах (паран метод и др.).

2.4.2. Обучение без учителя. Рассмотренный алгоритм обучения нейронной сети с помощью процедуры обратного распространения подразумевает наличие некоего внешнего звена, предоставляющего сети кроме входных также и целевые выходные образы.

Алгоритмы, пользующиеся подобной концепцией, называются алгоритмами обучения с учителем. Для их успешного функционирования необходимо наличие экспертов, создающих на предварительном этапе для каждого входного образа эталонный выходной. Так как создание искусственного интеллекта движется по пути копирования природных прообразов, ученые не прекращают спор на тему, можно ли считать алгоритмы обучения с учителем натуральными или же они полностью искусственны. Например, обучение человеческого мозга, на первый взгляд, происходит без учителя: на зрительные, слуховые, тактильные и прочие рецепторы поступает информация извне, и внутри нервной системы происходит некая самоорганизация. Однако нельзя отрицать и того, что в жизни человека не мало учителей — и в буквальном, и в переносном смысле, — которые координируют внешние воздействия. Вместе с тем, чем бы ни закончился спор приверженцев этих двух концепций обучения — с учителем и без учителя, они обе имеют право на существование.

Главная черта, делающая обучение без учителя привлекательным, — это его «самостоятельность». Процесс обучения, как и в случае обучения с учителем, заключается в подстраивании весов сети. Некоторые алгоритмы, правда, изменяют и структуру сети, т.е. количество нейронов и их взаимосвязи, но такие преобразования правильнее назвать более широким термином — самоорганизацией, и здесь они рассматриваться не будут. Очевидно, что подстройка весов может проводиться только на основании информации, доступной в нейроне, т.е. его состояния и уже имеющихся весовых коэффициентов. Исходя из этого соображения и, что более важно, по аналогии с известными принципами самоорганизации нервных клеток, построены алгоритмы обучения Хебба.

Сигнальный метод обучения Хебба заключается в изменении весов по следующему правилу:

$$w_{ij} := w_{ij} + \eta o_j^{(q-1)} o_i^{(q)},$$

где $o_j^{(q-1)}$ — выходное значение j -го нейрона слоя $(q-1)$, $o_i^{(q)}$ — выходное значение i -го нейрона слоя q ; w_{ij} — весовой коэффициент синапса, соединяющего эти нейроны, η — коэффициент скорости обучения. Здесь и далее, для общности, под q подразумевается произвольный слой сети. При обучении по данному методу усиливаются связи между возбужденными нейронами.

Полный алгоритм обучения с применением вышеприведенной формулы будет выглядеть так:

1. На стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения.

2. На входы сети подается входной образ, и сигналы возбуждения распространяются по всем слоям согласно принципам классических сетей прямого распространения (feedforward), т.е. для каждого нейрона рассчитывается взвешенная сумма его входов, к которой затем применяется активационная (передаточная) функция нейрона, в результате чего получается его выходное значение.

3. На основании полученных выходных значений нейронов по приведенной формуле производится изменение весовых коэффициентов.

4. Цикл с шага 2, пока выходные значения сети не стабилизируются с заданной точностью. Применение этого нового способа определения завершения обучения, отличного от использовавшегося для сети обратного распространения, обусловлено тем, что подстраиваемые значения синапсов фактически не ограничены.

На втором шаге цикла попеременно предъявляются все образы из входного набора.

Следует отметить, что вид откликов на каждый класс входных образов неизвестен заранее и будет представлять собой произвольное сочетание состояний нейронов выходного слоя, обусловленное случайным распределением весов на стадии инициализации. Вместе с тем, сеть способна обобщать схожие образы, относя их к одному классу. Тестирование обученной сети позволяет определить

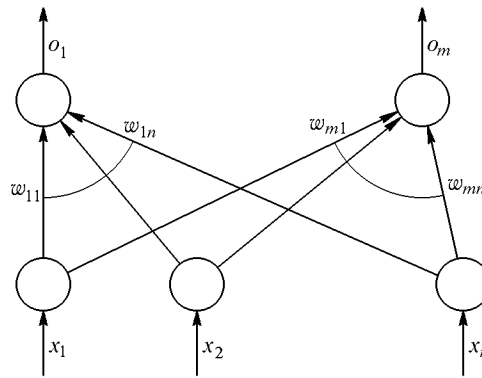


Рис. 2.10. Структура сети Кохонена

топологию классов в выходном слое. Для приведения откликов обученной сети к удобному представлению можно дополнить сеть одним слоем, который, например, по алгоритму обучения однослойного персептрона необходимо заставить отображать выходные реакции сети в требуемые образы.

Другой алгоритм обучения без учителя — алгоритм Кохонена (Kohonen) — предусматривает самообучение по правилу «победитель забирает все». Структура сети, реализующей данное правило, представлена на рис. 2.10.

Критерий подстройки весов сети (все векторы весов должны быть нормализованы, т.е. иметь единичную длину: $\|\mathbf{w}_i\| = 1$, $i = 1, 2, \dots, m$) выглядит следующим образом:

$$\|\mathbf{x} - \mathbf{w}_r\| = \min_{i=1,2,\dots,m} \|\mathbf{x} - \mathbf{w}_i\|,$$

где индекс r обозначает нейрон-победитель, соответствующий вектору весов $\mathbf{w}_r$, который ближе всех расположен к (текущему) входному вектору $\mathbf{x}$.

Поскольку (с учетом того, что $\mathbf{w}_i^T \mathbf{w}_i = \|\mathbf{w}_i\|^2 = 1$)

$$\begin{aligned} \|\mathbf{x} - \mathbf{w}_i\|^2 &= (\mathbf{x} - \mathbf{w}_i)^T (\mathbf{x} - \mathbf{w}_i) = \\ &= \mathbf{x}^T \mathbf{x} - 2\mathbf{w}_i^T \mathbf{x} + \mathbf{w}_i^T \mathbf{w}_i = \mathbf{x}^T \mathbf{x} - 2\mathbf{w}_i^T \mathbf{x} + 1, \end{aligned}$$

процедура нахождения $\mathbf{w}_r$ эквивалентна решению оптимизационной задачи

$$\mathbf{w}_r^T \mathbf{x} = \max_{i=1,2,\dots,m} \mathbf{w}_i^T \mathbf{x},$$

чему можно дать следующую геометрическую интерпретацию (см. рис. 2.11).

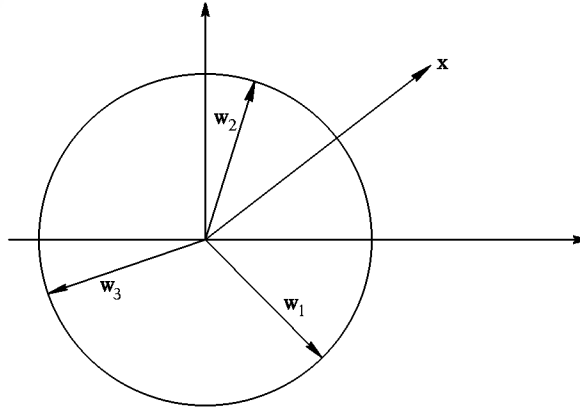


Рис. 2.11. Иллюстрация к алгоритму самообучения Кохонена

Так как скалярное произведение $\mathbf{w}_i^T \mathbf{x}$ с учетом $\|\mathbf{w}_i\| = 1$ представляет собой просто проекцию вектора $\mathbf{x}$ на направление вектора $\mathbf{w}_i$, то нейрон-победитель определяется по тому вектору весов, чье

направление ближе всего к направлению $\mathbf{x}$ (на рис. 2.11 таким является вектор $\mathbf{w}_2$).

После выявления нейрона-победителя его выход устанавливается равным единице (у остальных нейронов устанавливаются нулевые выходы), а веса корректируются так, чтобы уменьшить квадрат величины рассогласования $\|\mathbf{x} - \mathbf{w}_r\|^2$. При использовании градиентного подхода это приводит к следующей математической формулировке:

$$\mathbf{w}_r := \mathbf{w}_r - \eta \frac{d\|\mathbf{x} - \mathbf{w}_r\|^2}{d\mathbf{w}_r},$$

где, как и раньше, η — константа, определяющая скорость обучения.

Найдем производную в правой части последнего выражения:

$$\begin{aligned} \frac{d\|\mathbf{x} - \mathbf{w}_r\|^2}{d\mathbf{w}_r} &= \frac{d((\mathbf{x} - \mathbf{w}_r)^\top (\mathbf{x} - \mathbf{w}_r))}{d\mathbf{w}_r} = \\ &= \frac{d(\mathbf{x}^\top \mathbf{x} - 2\mathbf{w}_r^\top \mathbf{x} + \mathbf{w}_r^\top \mathbf{w}_r)}{d\mathbf{w}_r} = -2(\mathbf{x} - \mathbf{w}_r). \end{aligned}$$

При этом

$$\mathbf{w}_r := \mathbf{w}_r + \eta(\mathbf{x} - \mathbf{w}_r).$$

Заметим, что коррекции весов у других нейронов не производится.

Алгоритм обучения (без учителя) Кохонена может быть теперь описан следующим образом.

1. Задаются случайные нормализованные по длине векторы $\mathbf{w}_i$.
2. Начало цикла обучения; ввод очередного вектора входов $\mathbf{x}$.
3. Определение нейрона-победителя, корректировка вектора его весов и задание единичного выхода:

$$\mathbf{w}_r := \mathbf{w}_r + \eta(\mathbf{x} - \mathbf{w}_r), \quad o_r = 1.$$

4. Нормализация найденного вектора:

$$\mathbf{w}_r := \frac{\mathbf{w}_r}{\|\mathbf{w}_r\|}.$$

5. Задание значений для остальных нейронов:

$$\mathbf{w}_i := \mathbf{w}_i, \quad o_i = 0, \quad i \neq r.$$

6. Проверка выполнения правила останова (в качестве такого правила можно принять, например, стабилизацию векторов весов

на каких-то значениях); если оно не выполнено — продолжение цикла обучения (переходом к шагу 2), в противном случае — переход к шагу 7.

7. Конец процедуры обучения.

Заметим, что выражение для коррекции вектора весовых коэффициентов нейрона-победителя может быть представлено в форме

$$\mathbf{w}_r := \mathbf{w}_r + \eta(\mathbf{x} - \mathbf{w}_r) = (1 - \eta)\mathbf{w}_r + \eta\mathbf{x},$$

т.е. новое (скорректированное) значение данного вектора является взвешенной суммой старого значения (до коррекции) и предъ-

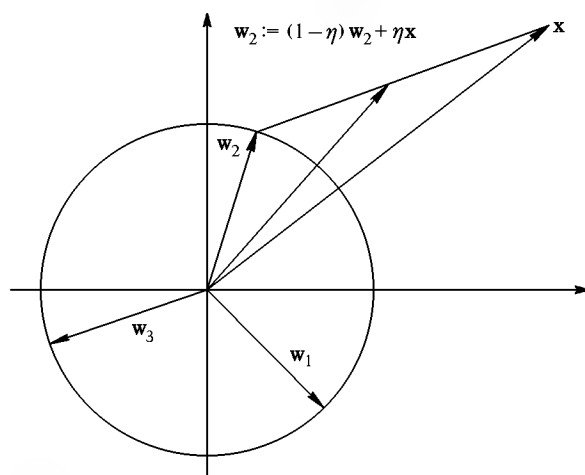


Рис. 2.12. Иллюстрация к процедуре коррекции вектора весов нейрона-победителя

явленного вектора входов, чему соответствует геометрическая иллюстрация рис. 2.12.

Нетрудно показать, что итоговым результатом подобных коррекций (в условиях рассматриваемого примера для двумерного случая) являются векторы весов, показывающие на центры кластеров (центры группирования) входных образов (рис. 2.13).

Иначе говоря, алгоритм обучения Кохонена обеспечивает решение задачи автоматической классификации, т.е. отнесения предъявленного вектора входов к одному из классов (на рис. 2.13 таких классов 3). Правда, такая классификация возможна только в случае, когда кластеры являются линейно разделимыми (гиперплоскостями) относительно начала координат в пространстве входов НС.

Отметим, что число нейронов НС для успешного решения указанной задачи должно быть не меньше, чем число кластеров; поскольку точное число кластеров может быть заранее неизвестно, количество нейронов задают с определенным запасом. «Лишние»

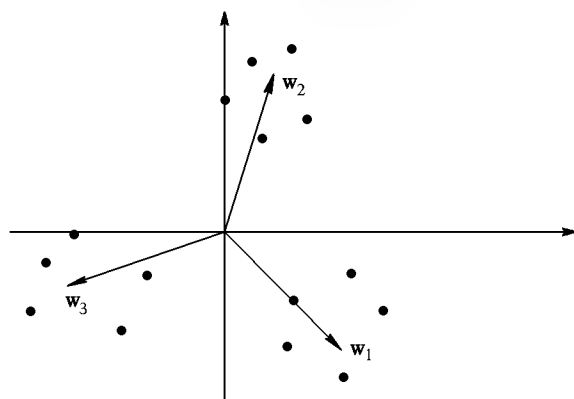


Рис. 2.13. Векторы весов НС после окончания процесса обучения

нейроны, у которых в процессе обучения сети веса изменяются хаотически по завершении данного процесса могут быть удалены.

2.5. Применение нейросети

После того как сеть обучена, мы можем применять ее для решения различных задач (рис. 2.14).

Важнейшая особенность человеческого мозга состоит в том, что, однажды обучившись определенному процессу, он может верно

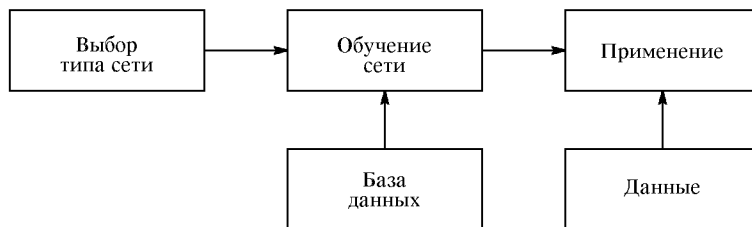


Рис. 2.14. Этапы нейросетевого проекта

действовать и в тех ситуациях, в которых он не бывал в процессе обучения. Например, мы можем читать почти любой почерк, даже

если видим его первый раз в жизни. Так же и нейросеть, грамотным образом обученная, может с большой вероятностью правильно реагировать на новые, не предъявленные ей ранее данные. Например, мы можем нарисовать букву «А» другим почерком, а затем предложить нашей сети классифицировать новое изображение.

Веса обученной сети хранят достаточно много информации о сходстве и различиях букв, поэтому можно рассчитывать на правильный ответ и для нового варианта изображения.

Области применения нейросетей: классификация. Отметим, что задачи классификации (типа распознавания букв) очень плохо алгоритмизируются. Если в случае распознавания букв верный ответ очевиден для нас заранее, то в более сложных практических задачах обученная нейросеть выступает как эксперт, обладающий большим опытом и способный дать ответ на трудный вопрос.

Примером такой задачи служит медицинская диагностика, где сеть может учитывать большое количество числовых параметров (энцефалограмма, давление, вес и т. д.). Конечно, «мнение» сети в этом случае нельзя считать окончательным. Классификация предприятий по степени их перспективности — это уже привычный способ использования нейросетей в практике западных компаний (деление компаний на перспективные и убыточные). При этом сеть также использует множество экономических показателей, сложным образом связанных между собой.

Нейросетевой подход особенно эффективен в задачах экспертной оценки по той причине, что он сочетает в себе способность компьютера к обработке чисел и способность мозга к обобщению и распознаванию. Говорят, что у хорошего врача способность к распознаванию в своей области столь велика, что он может провести приблизительную диагностику уже по внешнему виду пациента. Можно согласиться также, что опытный трейдер чувствует направление движения рынка по виду графика. Однако в первом случае все факторы наглядны, т. е. характеристики пациента мгновенно воспринимаются мозгом как «бледное лицо», «блеск в глазах» и т. д. Во втором же случае учитывается только один фактор, показанный на графике, — курс за определенный период времени. Нейросеть позволяет обрабатывать огромное количество факторов (до нескольких тысяч), независимо от их наглядности, — это универсальный «хороший врач», который может поставить свой диагноз в любой области.

Кластеризация и поиск зависимостей. Помимо задач классификации, нейросети широко используются для поиска зависимостей в данных и кластеризации.

Например, нейросеть на основе методики МГУА (метод группового учета аргументов) позволяет на основе обучающей выборки построить зависимость одного параметра от других в виде полинома. Такая сеть может не только мгновенно выучить таблицу умножения, но и найти сложные скрытые зависимости в данных (например, финансовых), которые не обнаруживаются стандартными статистическими методами.

Кластеризация — это разбиение набора примеров на несколько компактных областей (кластеров), причем число кластеров заранее неизвестно. Кластеризация позволяет представить неоднородные данные в более наглядном виде и использовать далее для исследования каждого кластера различные методы. Например, таким образом можно быстро выявить фальсифицированные страховые случаи или недобросовестные предприятия.

Прогнозирование. Задачи прогнозирования особенно важны для практики, в частности, для финансовых приложений, поэтому поясним способы применения нейросетей в этой области более подробно.

Рассмотрим практическую задачу, ответ в которой очевиден, — задачу прогнозирования курса акций на 1 день вперед.

Пусть у нас имеется база данных, содержащая значения курса за последние 300 дней. Простейший вариант в данном случае попытаться построить прогноз завтрашней цены на основе курсов за последние несколько дней. Понятно, что прогнозирующая сеть должна иметь всего один выход и столько входов, сколько предыдущих значений мы хотим использовать для прогноза — например, 4 последних значения. Составить обучающий пример очень просто: входными значениями будут курсы за 4 последовательных дня, а желаемым выходом — известный нам курс в следующий день за этими четырьмя, т. е. каждая строка таблицы с обучающей последовательностью (выборкой) представляет собой обучающий пример, где первые 4 числа — входные значения сети, а пятое число — желаемое значение выхода.

Заметим, что объем обучающей выборки зависит от выбранного количества входов. Если сделать 299 входов, то такая сеть потенциально могла бы строить лучший прогноз, чем сеть с 4 входами, однако в этом случае мы имеем дело с огромным массивом данных, что делает обучение и использование сети практически невозможным. При выборе числа входов следует учитывать это, выбирая разумный компромисс между глубиной предсказания (число входов) и качеством обучения (объем тренировочного набора).

Вообще говоря, в зависимости от типа решаемой задачи, целесообразно применять нейронную сеть наиболее подходящей для

такой задачи структуры. Рассмотрим некоторые, наиболее употребительные виды НС.

2.5.1. Персептроны. В качестве научного предмета искусственные нейронные сети впервые заявили о себе в 40-е годы. Стремясь воспроизвести функции человеческого мозга, исследователи создали простые аппаратные (а позже программные) модели биологического нейрона и системы его соединений. Когда нейрофизиологи достигли более глубокого понимания нервной системы человека, эти ранние попытки стали восприниматься как весьма грубые аппроксимации. Тем не менее на этом пути были достигнуты впечатляющие результаты, стимулировавшие дальнейшие исследования, приведшие к созданию более изощренных сетей.

Первое систематическое изучение искусственных нейронных сетей было предпринято Маккалохом и Питтсом в 1943 г. Позднее они исследовали сетевые парадигмы для распознавания изображений, подвергаемых сдвигам и поворотам, используя при этом простую нейронную модель, показанную на рис. 2.15. Элемент Σ умножает каждый вход x_i на вес w_i и суммирует взвешенные

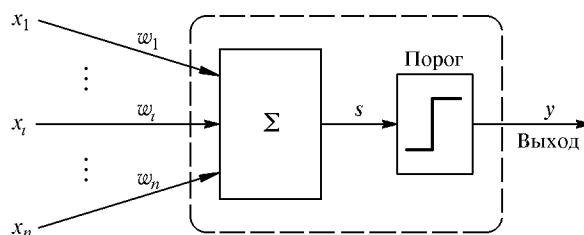


Рис. 2.15. Персептронный нейрон

входы. Если эта сумма больше заданного порогового значения, выход равен единице, в противном случае — нулю. Эти системы (и множество им подобных) получили название *персептронов*. Они состоят из одного слоя искусственных нейронов, соединенных с помощью весовых коэффициентов с множеством входов (см. рис. 2.16), хотя в принципе описываются и более сложные системы.

В 60-е годы персептроны вызвали большой интерес и оптимизм. Розенблатт доказал замечательные теоремы об обучении персептронов, приводимые ниже. Уидроу дал ряд убедительных демонстраций систем персептронного типа, и исследователи во всем мире стремились изучить возможности этих систем. Первоначальная эйфория сменилась разочарованием, когда оказалось,

что перцептроны не способны обучиться решению ряда простых задач. Минский строго проанализировал эту проблему и показал, что имеются жесткие ограничения на то, что могут выполнять однослойные перцептроны, и, следовательно, на то, чему они могут обучаться. Так как в то время методы обучения многослойных сетей не были известны, исследователи перешли в более многообещающие области, и исследования в области нейронных сетей

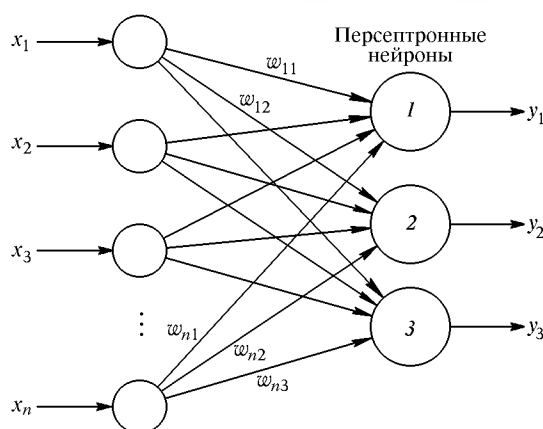


Рис. 2.16. Перцептрон со многими выходами

пришли в упадок. Недавнее открытие методов обучения многослойных сетей в большей степени, чем какой-либо иной фактор, повлияло на возрождение интереса и исследовательских усилий.

Работа Минского возможно и охладила пыл энтузиастов перцептрона, но обеспечила время для необходимой консолидации и развития лежащей в основе теории. Важно отметить, что анализ Минского не был опровергнут. Он остается важным исследованием и должен изучаться, чтобы ошибки 60-х годов не повторились.

Несмотря на свои ограничения, перцептроны широко изучались (хотя не слишком широко использовались). Теория перцептронов является основой для многих других типов искусственных нейронных сетей, в силу чего они являются логической исходной точкой для изучения искусственных нейронных сетей.

Рассмотрим в качестве примера трехнейронный перцептрон (рис. 2.16), нейроны которого имеют активационную функцию в виде единичного скачка.

На n входов поступают входные сигналы, проходящие по синапсам, на три нейрона, образующие единственный слой этой сети

и выдающие три выходных сигнала:

$$y_j = f \left(\sum_{i=1}^n x_i w_{ij} \right), \quad j = 1, \dots, 3.$$

Очевидно, что все весовые коэффициенты синапсов одного слоя нейронов можно свести в матрицу $\mathbf{W}$, в которой каждый элемент w_{ij} задает величину i -й синаптической связи j -го нейрона. Таким образом, процесс, происходящий в нейронной сети, может быть записан в матричной форме:

$$\mathbf{Y} = \mathbf{f}(\mathbf{XW}),$$

где $\mathbf{X}$ и $\mathbf{Y}$ — соответственно входной и выходной сигнальные векторы (здесь и далее под вектором понимается вектор-строка), $\mathbf{f}(\mathbf{S})$ — активационная функция, применяемая поэлементно к компонентам вектора $\mathbf{S}$.

На рис. 2.17 представлен двухслойный персептрон, полученный из персептрона с рис. 2.16 путем добавления второго слоя, состоящего из двух нейронов.

Здесь уместно отметить важную роль нелинейности активационной функции, так как, если бы она не обладала данным свойст-

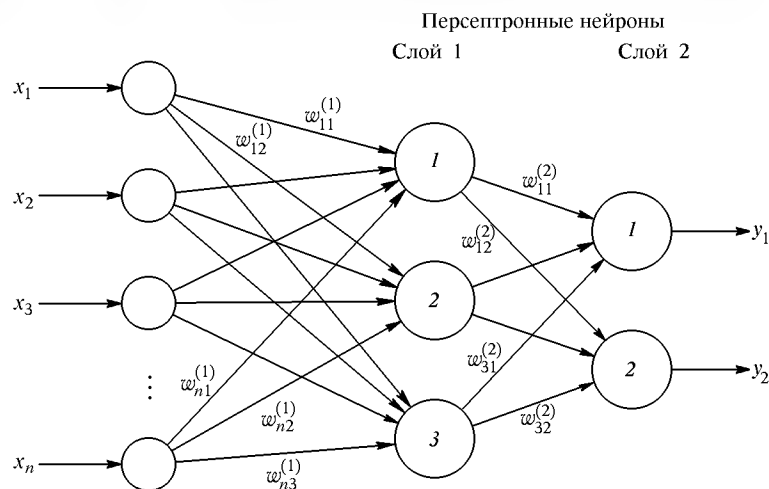


Рис. 2.17. Двухслойный персептрон

вом или не входила в алгоритм работы каждого нейрона, результат функционирования любой Q -слойной нейронной сети с весовыми

матрицами $\mathbf{W}^{(q)}$ для каждого слоя $q = 1, \dots, Q$ сводился бы к перемножению входного вектора сигналов $\mathbf{X}$ на матрицу:

$$\mathbf{W}_{(\Sigma)} = \mathbf{W}^{(1)} \dots \mathbf{W}^{(q)} \dots \mathbf{W}^{(Q)}.$$

Фактически такая Q -слойная нейронная сеть эквивалентна сети с одним скрытым слоем и с весовой матрицей единственного слоя $\mathbf{W}_{(\Sigma)}$:

$$\mathbf{Y} = \mathbf{XW}_{(\Sigma)}.$$

Работа персептрона сводится к классификации (обобщению) входных сигналов, принадлежащих n -мерному гиперпространству, по некоторому числу классов. С математической точки зрения это происходит путем разбиения гиперпространства гиперплоскостями. Для случая однослойного персептрона

$$\sum_{i=1}^n x_i w_{ij} = \theta_j, \quad j = 1, 2, \dots, m.$$

Каждая полученная область является областью определения отдельного класса. Число таких классов для персептрона не превышает 2^n , где n — число его входов. Однако не все из классов могут быть разделены данной нейронной сетью. Например, однослойный персептрон, состоящий из одного нейрона с двумя входами, не может реализовать логическую функцию ИСКЛЮЧАЮЩЕЕ ИЛИ, т.е. не способен разделить плоскость (двумерное гиперпространство) на две полуплоскости так, чтобы осуществить классификацию входных сигналов по классам А и В (см. табл. 2.3).

Уравнение сети для этого случая

$$x_1 w_1 + x_2 w_2 = \theta$$

является уравнением прямой (одномерной гиперплоскости), которая ни при каких условиях не может разделить плоскость так, чтобы точки из множества входных сигналов, принадлежащие разным классам, оказались по разные стороны от прямой (рис. 2.18). Невозможность реализации однослойным персептроном этой функции получила название проблемы ИСКЛЮЧАЮЩЕГО ИЛИ.

Отметим, что функции, которые не реализуются однослойным персептроном, называются линейно неразделимыми. Решение задач, подпадающих под это ограничение, заключается в применении двух- и более слойных сетей или сетей с нелинейными синапсами, однако и тогда существует вероятность, что корректное разделение некоторых входных сигналов на классы невозможно.

Обучение персептрона сводится к формированию весов связей между первым и вторым (см. рис. 2.17) слоями в соответствии со следующим алгоритмом.

Таблица 2.3. Логическая функция
ИСКЛЮЧАЮЩЕЕ ИЛИ

x_2	0	1
x_1		
0	В	А
1	А	В

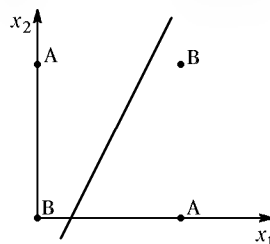


Рис. 2.18. Линейная неразделимость функции ИСКЛЮЧАЮЩЕЕ ИЛИ

Шаг 1. Проинициализировать элементы весовой матрицы (обычно небольшими случайными значениями).

Шаг 2. Подать на входы один из входных векторов, которые сеть должна научиться различать, и вычислить ее выход.

Шаг 3. Если выход правильный, перейти на шаг 4.

Иначе — вычислить разницу между идеальным d и полученным Y значениями выхода:

$$\delta = d - Y.$$

Модифицировать веса в соответствии с формулой

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta x_i,$$

где t и $(t+1)$ — номера соответственно текущей и следующей итераций; η — коэффициент скорости обучения, $0 < \eta < 1$; i — номер входа; j — номер нейрона в слое.

Очевидно, что если $d > Y$, то весовые коэффициенты будут увеличены и, тем самым, уменьшат ошибку. В противном случае они будут уменьшены, и Y тоже уменьшится, приближаясь к d .

Шаг 4. Цикл с шага 2, пока сеть не перестанет ошибаться.

На втором шаге на разных итерациях поочередно в случайном порядке предъявляются все возможные входные векторы. К сожалению, нельзя заранее определить число итераций, которые потребуется выполнить, а в некоторых случаях и гарантировать полный успех.

Сходимость рассмотренной процедуры устанавливается теоремами, утверждающими, что для любой классификации обучающей последовательности можно подобрать такой набор (из бесконечного набора) элементарных нейронов, в котором будет осуществлено разделение обучающей последовательности при помощи линейного

решающего правила, и что, если относительно задуманной классификации можно найти набор элементов, в котором существует решение, то в рамках этого набора оно будет достигнуто в конечный промежуток времени.

2.5.2. Нейронные сети встречного распространения. Объединение разнотипных нейронных структур в единой архитектуре зачастую приводит к свойствам, которых нет у них по отдельности. Причем именно каскадные соединения нейронных структур, специализирующихся на решении различных задач, позволяют решить проблему комплексно.

Нейронные сети встречного распространения, состоящие из входного слоя нейронов и так называемых слоев нейронов Кохонена и Гроссберга, по своим характеристикам существенно превосходят возможности сетей с одним скрытым слоем нейронов. Так, время их обучения задачам распознавания и кластеризации более, чем в сто раз меньше времени обучения аналогичным задачам сетей с обратным распространением. Это может быть полезно в тех приложениях, где долгая обучающая процедура невозможна.

Одними из определяющих характеристик сети встречного распространения являются ее хорошие способности к обобщению, позволяющие получать правильный выход даже при неполном или зашумленном входном векторе. Это позволяет эффективно использовать данную сеть для распознавания и восстановления образов, а также для усиления сигналов.

В процессе обучения сети встречного распространения входные векторы ассоциируются с соответствующими выходными векторами. Эти векторы могут быть двоичными или непрерывными. После обучения сеть формирует выходные сигналы, соответствующие входным сигналам. Обобщающая способность сети дает возможность получать правильный выход, когда входной вектор неполон или искажен.

Сеть встречного распространения имеет два слоя с последовательными связями. Первый слой — слой Кохонена, второй — слой Гроссберга. Каждый элемент входного сигнала подается на все нейроны слоя Кохонена. Каждый нейрон слоя Кохонена соединен со всеми нейронами слоя Гроссберга. Отличие сети встречного распространения от других многослойных сетей с последовательными связями состоит в операциях, выполняемых нейронами Кохонена и Гроссберга.

В режиме функционирования сети предъявляется входной сигнал X и формируется выходной сигнал Y . В режиме обучения на вход сети подается входной сигнал и веса корректируются, чтобы сеть выдавала требуемый выходной сигнал.

Функционирование сети

Слой Кохонена. В своей простейшей форме слой Кохонена функционирует по правилу «победитель получает все». Для данного входного вектора один и только один нейрон Кохонена выдает логическую единицу, все остальные выдают ноль. Выход каждого нейрона Кохонена является просто суммой взвешенных элементов входных сигналов:

$$s_j = w_{1j}x_1 + w_{2j}x_2 + \dots + w_{nj}x_n = \sum_i x_i w_{ij},$$

где s_j — выход j -го нейрона Кохонена, $\mathbf{W}_j = (w_{1j}, w_{2j}, \dots, w_{nj})$ — вектор весов j -го нейрона Кохонена, $\mathbf{X} = (x_1, x_2, \dots, x_n)$ — вектор входного сигнала, или в векторно-матричной форме:

$$\mathbf{S} = \mathbf{XW},$$

где $\mathbf{S}$ — вектор выходов слоя Кохонена.

Нейрон Кохонена с максимальным значением s_j является «победителем». Его выход равен единице, у остальных он равен нулю.

Слой Гроссберга. Слой Гроссберга функционирует в сходной манере. Его выход является взвешенной суммой выходов слоя Кохонена (т.е. он является слоем нейронов с линейными активационными функциями).

Если слой Кохонена функционирует таким образом, что лишь один выход равен единице, а остальные равны нулю, то каждый нейрон слоя Гроссберга выдает величину веса, который связывает этот нейрон с единственным нейроном Кохонена, чей выход отличен от нуля.

Предварительная обработка входных сигналов. Рассматриваемая НС требует предварительной обработки входных векторов путем их нормализации. Такая нормализация выполняется путем деления каждой компоненты входного вектора на длину вектора (квадратный корень из суммы квадратов компонент вектора). Это превращает входной вектор в единичный вектор с тем же направлением, т.е. в вектор единичной длины в n -мерном пространстве.

Обучение слоя Кохонена. Слой Кохонена классифицирует входные векторы в группы схожих. Это достигается с помощью такой подстройки весов слоя Кохонена, что близкие входные векторы активируют один и тот же нейрон данного слоя (затем задачей слоя Гроссберга является получение требуемых выходов).

Слой Кохонена обучается без учителя (самообучается). В результате обучения слой приобретает способность разделять несхожие входные векторы. Какой именно нейрон будет активироваться при предъявлении конкретного входного сигнала, заранее трудно предсказать.

При обучении слоя Кохонена на вход подается входной вектор и вычисляются его скалярные произведения с векторами весов всех нейронов. Скалярное произведение является мерой сходства между входным вектором и вектором весов. Нейрон с максимальным значением скалярного произведения объявляется «победителем» и его веса подстраиваются (весовой вектор приближается к входному).

Уравнение, описывающее процесс обучения, имеет вид

$$w_n = w_c + \eta(x - w_c),$$

где w_n — новое значение веса, соединяющего входную компоненту x с выигравшим нейроном, w_c — предыдущее значение этого веса, η — коэффициент скорости обучения.

Каждый вес, связанный с выигравшим нейроном Кохонена, изменяется пропорционально разности между его величиной и величиной входа, к которому он присоединен. Направление изменения минимизирует разность между весом и соответствующим элементом входного сигнала.

Коэффициент скорости обучения η вначале обычно полагается равным 0,7 и может затем постепенно уменьшаться в процессе обучения. Это позволяет делать большие начальные шаги для быстрого грубого обучения и меньшие шаги при подходе к окончательной величине.

Если бы с каждым нейроном Кохонена ассоциировался один входной вектор, то слой Кохонена мог бы быть обучен с помощью одного вычисления на вес ($\eta = 1$). Как правило, обучающее множество включает много сходных между собой входных векторов, и сеть должна быть обучена активировать один и тот же нейрон Кохонена для каждого из них. Веса этого нейрона должны получаться усреднением входных векторов, которые должны его активировать.

Обучение слоя Гроссберга. Выходы слоя Кохонена подаются на входы нейронов слоя Гроссберга. Выходы нейронов вычисляются, как при обычном функционировании. Далее каждый вес корректируется лишь в том случае, если он соединен с нейроном Кохонена, имеющим ненулевой выход. Величина коррекции веса пропорциональна разности между весом и требуемым выходом нейрона Гроссберга.

Обучение слоя Гроссберга — это обучение с учителем, алгоритм использует заданные желаемые выходы.

В полной модели сети встречного распространения имеется возможность получать выходные сигналы по входным и наоборот. Этим двум действиям соответствуют прямое и обратное распространение сигналов.

Области применения: распознавание образов, восстановление образов (ассоциативная память), сжатие данных (с потерями).

Недостатки. Сеть не дает возможности строить точные аппроксимации (точные отображения). В этом сеть значительно уступает сетям с обратным распространением ошибки.

К недостаткам модели также следует отнести слабую теоретическую проработку модификаций сети встречного распространения.

Преимущества. Сеть встречного распространения проста. Она дает возможность извлекать статистические свойства из множеств входных сигналов. Кохоненом показано, что для полностью обученной сети вероятность того, что случайно выбранный входной вектор (в соответствии с функцией плотности вероятности входного множества) будет ближайшим к любому заданному весовому вектору, равна $1/p$, p — число нейронов Кохонена.

Сеть быстро обучается. Время обучения по сравнению с обратным распространением может быть в 100 раз меньше. По своим возможностям строить отображения сеть встречного распространения значительно превосходит однослойные перцептроны.

Сеть полезна для приложений, в которых требуется быстрая начальная аппроксимация.

Сеть дает возможность строить функцию и обратную к ней, что находит применение при решении практических задач.

Модификации. Сети встречного распространения могут различаться способами определения начальных значений синаптических весов. Так, кроме традиционных случайных значений из заданного диапазона, могут быть использованы значения в соответствии с известным методом выпуклой комбинации.

Для повышения эффективности обучения применяется добавление шума к входным векторам.

Еще один метод повышения эффективности обучения — наделение каждого нейрона «чувством справедливости». Если нейрон становится победителем чаще, чем $1/p$ (p — число нейронов Кохонена), то ему временно увеличивают порог, давая тем самым обучаться и другим нейронам.

Кроме «метода аккредитации», при котором для каждого входного вектора активируется лишь один нейрон Кохонена, может быть использован «метод интерполяции», при использовании которого целая группа нейронов Кохонена, имеющих наибольшие выходы, может передавать свои выходные сигналы в слой Гроссберга. Этот метод повышает точность отображений, реализуемых сетью, и реализован в так называемых *самоорганизующихся картах*.

2.5.3. Нейронные сети Хопфилда и Хэмминга. Среди различных конфигураций искусственных нейронных сетей (НС) встречаются такие, при классификации которых по принципу обучения, строго говоря, не подходят ни обучение с учителем, ни обучение без учителя. В таких сетях весовые коэффициенты синапсов рассчитываются только однажды перед началом функционирования сети на основе информации об обрабатываемых данных, и все обучение сети сводится именно к этому расчету. С одной стороны, предъявление априорной информации можно расценивать как помощь учителя, но с другой, — сеть фактически просто запоминает образцы до того, как на ее вход поступают реальные данные, и не может изменять свое поведение, поэтому говорить о звене обратной связи с «миром» (учителем) не приходится. Из сетей с подобной логикой работы наиболее известны *сеть Хопфилда* и *сеть Хэмминга* (представляющие собой разновидности сетей с обратными связями), которые обычно используются для организации ассоциативной памяти. Далее речь пойдет именно о них. Структурная схема сети Хопфилда приведена на рис. 2.19. Она состоит

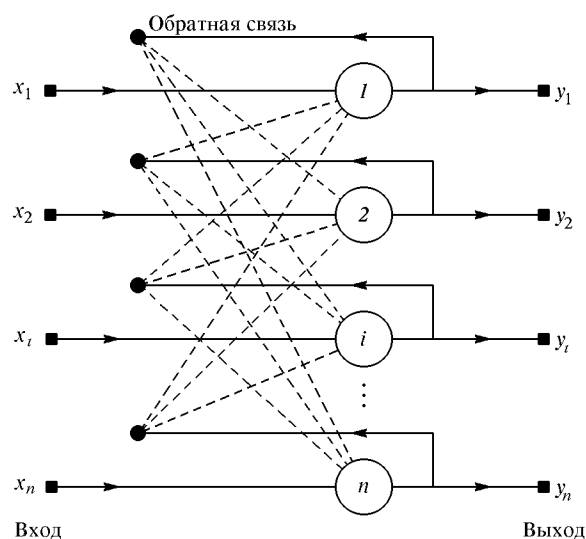


Рис. 2.19. Структурная схема сети Хопфилда

из единственного слоя нейронов, число которых является одновременно числом входов и выходов сети. Каждый нейрон связан синапсами со всеми остальными нейронами, а также имеет один

входной синапс, через который осуществляется ввод сигнала. Выходные сигналы, как обычно, образуются на аксонах.

Задача, решаемая данной сетью в качестве ассоциативной памяти, как правило, формулируется следующим образом. Известен некоторый набор двоичных сигналов (изображений, звуковых оцифровок, прочих данных, описывающих некие объекты или характеристики процессов), которые считаются образцовыми. Сеть должна уметь из произвольного неидеального сигнала, поданного на ее вход, выделить («вспомнить» по частичной информации) соответствующий образец (если такой есть) или «дать заключение» о том, что входные данные не соответствуют ни одному из образцов. В общем случае, любой сигнал может быть описан вектором $\mathbf{X} = \{x_i : i = 1, 2, \dots, n\}$, n — число нейронов в сети и размерность входных и выходных векторов. Каждый элемент x_i равен либо $+1$, либо -1 . Обозначим вектор, описывающий k -й образец, через $\mathbf{X}^k$, а его компоненты, соответственно, x_i^k , $k = 1, 2, \dots, m$, где m — в данном случае число образцов. Когда сеть распознает (или «вспомнит») какой-либо образец на основе предъявленных ей данных, ее выходы будут содержать именно его, т.е. $\mathbf{Y} = \mathbf{X}^k$, где $\mathbf{Y}$ — вектор выходных значений сети: $\mathbf{Y} = \{y_i : i = 1, 2, \dots, n\}$. В противном случае, выходной вектор не совпадет ни с одним образцовым.

Если, например, сигналы представляют собой некие изображения, то, отобразив в графическом виде данные с выхода сети, можно будет увидеть картинку, полностью совпадающую с одной из образцовых (в случае успеха), или же «вольную импровизацию» сети (в случае неудачи).

На стадии инициализации сети весовые коэффициенты синапсов устанавливаются следующим образом:

$$w_{ij} = \begin{cases} \sum_{k=1}^m x_i^k x_j^k, & i \neq j, \\ 0, & i = j. \end{cases}$$

Здесь i и j — индексы, соответственно, предсинаптического и постсинаптического нейронов; x_i^k, x_j^k — i -й и j -й элементы вектора k -го образца.

Алгоритм функционирования сети следующий (t — номер итерации):

1. На входы сети подается неизвестный сигнал. Фактически его ввод осуществляется непосредственной установкой значений аксонов:

$$y_i(0) = x_i, \quad i = 1, 2, \dots, n,$$

поэтому обозначение на схеме сети входных синапсов в явном виде носит чисто условный характер. Ноль в скобке справа от y_i означает нулевую итерацию в цикле работы сети.

2. Рассчитывается новое состояние нейронов

$$s_j(t+1) = \sum_{i=1}^n w_{ij} y_i(t), \quad j = 1, 2, \dots, n,$$

и новые значения аксонов

$$y_j(t+1) = f[s_j(t+1)],$$

где f — пороговая активационная функция с порогом $\theta = 0$ (см. табл. 2.1).

3. Проверка, изменились ли выходные значения аксонов за последнюю итерацию. Если да — переход к пункту 2, иначе (если выходы застabilизировались) — конец. При этом выходной вектор представляет собой образец, наилучшим образом сочетающийся с входными данными.

Таким образом, когда подается новый вектор, сеть переходит из вершины в вершину, пока не стабилизируется. Устойчивая вершина определяется сетевыми весами и текущими входами. Если входной вектор частично неправилен или неполон, сеть стабилизируется в вершине, ближайшей к желаемой.

Показано, что достаточным условием устойчивой работы такой сети является выполнение условий

$$w_{ij} = w_{ji}, \quad w_{ii} = 0.$$

Как говорилось выше, иногда сеть не может провести распознавание и выдает на выходе несуществующий образ. Это связано с проблемой ограниченности возможностей сети. Для сети Хопфилда число запоминаемых образов N не должно превышать величины, примерно равной $0,15n$. Кроме того, если два образа A и B сильно похожи, они, возможно, будут вызывать у сети перекрестные ассоциации, т. е. предъявление на входы сети вектора A приведет к появлению на ее выходах вектора B и наоборот.

Когда нет необходимости, чтобы сеть в явном виде выдавала образец, т. е. достаточно, скажем, получать номер образца, ассоциативную память успешно реализует сеть Хэмминга. Данная сеть характеризуется, по сравнению с сетью Хопфилда, меньшими затратами на память и объемом вычислений, что становится очевидным из ее структуры (рис. 2.20).

Сеть состоит из двух слоев. Первый и второй слои имеют по m нейронов, где m — число образцов. Нейроны первого слоя

имеют по n синапсов, соединенных с входами сети (образующими фиктивный нулевой слой). Нейроны второго слоя связаны между собой ингибиторными (отрицательными обратными) синаптическими связями. Единственный синапс с положительной обратной связью для каждого нейрона соединен с его же аксоном.

Идея работы сети состоит в нахождении расстояния Хэмминга от тестируемого образа до всех образов (расстоянием Хэмминга называется число отличающихся битов в двух бинарных векторах). Сеть должна выбрать образец с минимальным расстоянием

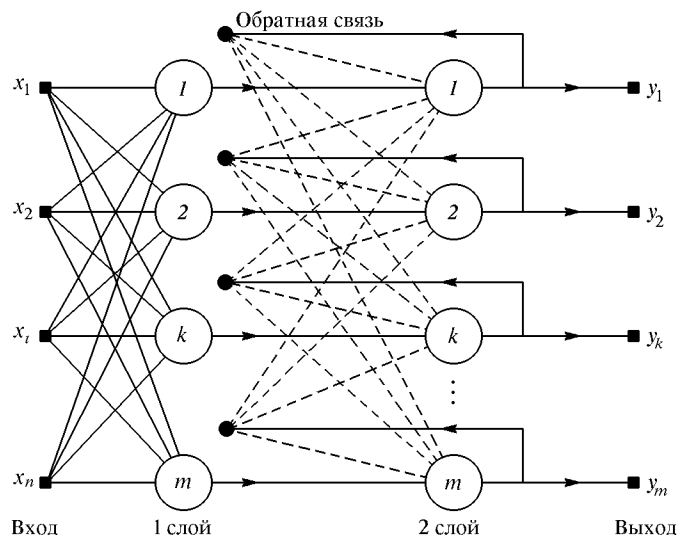


Рис. 2.20. Структурная схема сети Хэмминга

Хэмминга до неизвестного входного сигнала, в результате чего будет активизирован только один выход сети, соответствующий этому образцу.

На стадии инициализации весовым коэффициентам первого слоя и порогу активационной функции присваиваются следующие значения:

$$w_{ik} = \frac{x_i^k}{2}, \quad \theta_k = \frac{n}{2}, \quad i = 1, 2, \dots, n, \quad k = 1, 2, \dots, m.$$

Здесь x_i^k — i -й элемент k -го образца.

Весовые коэффициенты тормозящих синапсов во втором слое берут равными некоторой величине $0 < \varepsilon < 1/m$. Синапс нейрона, связанный с его же аксоном, имеет вес $+1$.

Алгоритм функционирования сети Хэмминга следующий:

1. На входы сети подается неизвестный вектор $\mathbf{X} = \{x_i : i = 1, \dots, n\}$, исходя из которого рассчитываются состояния нейронов первого слоя (верхний индекс в скобках указывает номер слоя):

$$y_j^{(1)} = s_j^{(1)} = \sum_{i=1}^n w_{ij} x_i + \theta_j, \quad j = 1, 2, \dots, m.$$

После этого полученными значениями инициализируются значения аксонов второго слоя:

$$y_j^{(2)} = y_j^{(1)}, \quad j = 1, 2, \dots, m.$$

2. Вычисляются новые состояния нейронов второго слоя:

$$s_j^{(2)}(t+1) = y_j(t) - \varepsilon \sum_{k=1}^m y_k^{(2)}(t), \quad k \neq j, \quad j = 1, 2, \dots, m,$$

и значения их аксонов:

$$y_j^{(2)}(t+1) = f[s_j^{(2)}(t+1)], \quad j = 1, 2, \dots, m.$$

3. Проверяется, изменились ли выходы нейронов второго слоя за последнюю итерацию. Если да — перейти к шагу 2. Иначе конец.

Из оценки алгоритма видно, что роль первого слоя весьма условна: воспользовавшись один раз на шаге 1 значениями его весовых коэффициентов, сеть больше не обращается к нему, поэтому первый слой может быть вообще исключен из сети (заменен на матрицу весовых коэффициентов).

В заключение можно сделать следующее обобщение. Сети Хопфилда и Хэмминга позволяют просто и эффективно разрешить задачу воссоздания образов по неполной и искаженной информации. Невысокая емкость сетей (число запоминаемых образов) объясняется тем, что сети не просто запоминают образы, а позволяют проводить их обобщение например, с помощью сети Хэмминга возможна классификация по критерию максимального правдоподобия. Вместе с тем, легкость построения программных и аппаратных моделей делают эти сети привлекательными для многих применений.

2.5.4. Сеть с радиальными базисными элементами (RBF). В общем случае под термином Radial Basis Function Network (сеть

РBF) понимается двухслойная сеть без обратных связей, которая содержит скрытый слой радиально симметричных скрытых нейронов (шаблонный слой). Для того чтобы шаблонный слой был радиально-симметричным, необходимо выполнение следующих условий:

- наличие центра, представленного в виде вектора во входном пространстве; обычно этот вектор сохраняется в пространстве весов от входного слоя к слою шаблонов;
- наличие способа измерения расстояния входного вектора от центра; обычно это стандартное евклидово расстояние;
- наличие специальной функции прохождения от одного аргумента, которая определяет выходной сигнал нейрона путем отображения функции расстояния; обычно используется функция Гаусса

$$\varphi(s) = e^{-s^2}.$$

Другими словами, выходной сигнал шаблонного нейрона это функция только от расстояния между входным вектором $\mathbf{X}$ и сохраненным центром $\mathbf{C}$:

$$f(\mathbf{X}) = \varphi\left(\frac{\|\mathbf{X} - \mathbf{C}\|}{\sigma}\right).$$

Выходной слой сети является линейным, так что выходы сети определяются выражением

$$y_j = \sum_{i=1}^K w_{ij} \varphi\left(\frac{\|\mathbf{X} - \mathbf{C}_i\|}{\sigma_i}\right), \quad j = 1, 2, \dots, m,$$

где $\mathbf{C}_i$ — центры, σ_i — отклонения радиальных элементов.

Обучение RBF-сети происходит в несколько этапов. Сначала определяются центры и отклонения для радиальных элементов; после этого оптимизируются параметры w_{ij} линейного выходного слоя.

Расположение центров должно соответствовать кластерам, реально присутствующим в исходных данных. Рассмотрим два наиболее часто используемых метода.

Выборка из выборки. В качестве центров радиальных элементов берутся несколько случайно выбранных точек обучающего множества. В силу случайности выбора они «представляют» распределение обучающих данных в статистическом смысле. Однако, если число радиальных элементов невелико, такое представление может быть неудовлетворительным.

Алгоритм *K*-средних. Этот алгоритм стремится выбрать оптимальное множество точек, являющихся центроидами кластеров в обучающих данных. При *K* радиальных элементах их центры располагаются таким образом, чтобы:

- каждая обучающая точка «относилась» к одному центру кластера и лежала к нему ближе, чем к любому другому центру;
- каждый центр кластера был центроидом множества обучающих точек, относящихся к этому кластеру.

После того как определено расположение центров, нужно найти отклонения. Величина отклонения (ее также называют сглаживающим фактором) определяет, насколько «острой» будет гауссова функция. Если эти функции выбраны слишком острыми, сеть не будет интерполировать данные между известными точками и потеряет способность к обобщению. Если же гауссовы функции взяты чересчур широкими, сеть не будет воспринимать мелкие детали (на самом деле сказанное — еще одна форма проявления дилеммы пере/недообучения). Как правило, отклонения выбираются таким образом, чтобы «колпак» каждой гауссовой функций захватывал несколько соседних центров. Для этого имеется несколько методов:

Явный. Отклонения задаются пользователем.

Изотропный. Отклонение берется одинаковым для всех элементов и определяется эвристически с учетом количества радиальных элементов и объема покрываемого пространства.

K ближайших соседей. Отклонение каждого элемента устанавливается (индивидуально) равным среднему расстоянию до его *K* ближайших соседей. Тем самым отклонения будут меньше в тех частях пространства, где точки расположены густо, — здесь будут хорошо учитываться детали, а там, где точек мало, отклонения будут большими (и будет производиться интерполяция).

После того как выбраны центры и отклонения, параметры выходного слоя оптимизируются с помощью стандартного метода линейной оптимизации — алгоритма псевдообратных матриц (сингулярного разложения).

Могут быть построены различные гибридные разновидности сетей с радиальными базисными функциями. Например, выходной слой может иметь нелинейные функции активации, и тогда для его обучения используется какой-либо из алгоритмов обучения многослойных сетей, например метод обратного распространения. Можно также обучать радиальный (скрытый) слой с помощью алгоритма обучения сети Кохонена — это еще один способ разместить центры так, чтобы они отражали расположение данных.

Сети RBF имеют ряд *преимуществ* перед рассмотренными многослойными сетями прямого распространения (хотя их структура и соответствует приведенной на рис. 2.5). Во-первых, они моделируют произвольную нелинейную функцию с помощью всего одного промежуточного слоя, тем самым избавляя нас от необходимости решать вопрос о числе слоев. Во-вторых, параметры линейной комбинации в выходном слое можно полностью оптимизировать с помощью хорошо известных методов линейной оптимизации, которые работают быстро и не испытывают трудностей с локальными минимумами, так мешающими при обучении с использованием алгоритма обратного распространения ошибки. Поэтому сеть RBF обучается очень быстро (на порядок быстрее, чем с использованием алгоритма обратного распространения).

Недостатки сетей RBF: данные сети обладают плохими экстраполирующими свойствами и получаются весьма громоздкими при большой размерности вектора входов.

2.5.5. Вероятностная нейронная сеть (PNN). Задача оценки плотности вероятности по имеющимся данным имеет давнюю историю в математической статистике.

Обычно при этом предполагается, что плотность имеет некоторый определенный вид (чаще всего, — что она имеет нормальное распределение). После этого оцениваются параметры модели. Нормальное распределение часто используется потому, что тогда параметры модели (среднее и стандартное отклонение) можно оценить аналитически.

Заметим, что предположение о нормальности далеко не всегда оправдано.

Другой подход к оценке плотности вероятности основан на так называемых *ядерных оценках*. Можно рассуждать так: тот факт, что наблюдение расположено в данной точке пространства, свидетельствует о том, что в этой точке имеется некоторая плотность вероятности. Кластеры из близко лежащих точек указывают на то, что в этом месте плотность вероятности большая. Вблизи наблюдения имеется большее доверие к уровню плотности, а по мере отдаления от него доверие убывает и стремится к нулю. В методе ядерных оценок в точке, соответствующей каждому наблюдению, помещается некоторая простая функция, затем все они складываются и в результате получается оценка для общей плотности вероятности. Чаще всего в качестве ядерных функций берутся гауссовы функции (с формой колокола). Если обучающих примеров достаточно количество, то такой метод дает достаточно хорошее приближение к истинной плотности вероятности.

Метод аппроксимации плотности вероятности с помощью ядерных функций во многом похож на метод радиальных базисных функций, и таким образом мы естественно приходим к понятиям вероятностной нейронной сети (PNN) и обобщенно-регрессионной нейронной сети (GRNN). PNN-сети предназначены для задач классификации, а GRNN для задач регрессии. Сети этих двух типов представляют собой реализацию методов ядерной аппроксимации, оформленных в виде нейронной сети.

Сеть PNN имеет по меньшей мере три слоя: входной, радиальный и выходной. Радиальные элементы берутся по одному на каждое обучающее наблюдение. Каждый из них представляет гауссову функцию с центром в этом наблюдении. Каждому классу соответствует один выходной элемент. Каждый такой элемент соединен со всеми радиальными элементами, относящимися к его классу, а со всеми остальными радиальными элементами он имеет нулевое соединение. Таким образом, выходной элемент просто складывает отклики всех элементов, принадлежащих к его классу. Значения выходных сигналов получаются пропорциональными ядерным оценкам вероятности принадлежности соответствующим классам, и, пронормировав их на единицу, мы получаем окончательные оценки вероятности принадлежности классам.

Выход рассматриваемой сети, соответствующий какому-либо классу, описывается выражением

$$y = \frac{1}{N\sigma^n} \sum_{k=1}^N \varphi\left(\frac{\|\mathbf{X} - \mathbf{X}^k\|}{\sigma}\right),$$

где n — размерность входного вектора, N — объем обучающей выборки, $\mathbf{X}^k$ — элемент (вектор) этой выборки, соответствующий отмеченному классу.

Базовая модель PNN-сети может иметь две модификации.

Вероятностная нейронная сеть имеет единственный управляющий параметр обучения, значение которого должно выбираться пользователем, — отклонение гауссовой функции σ (параметр сглаживания). Как и в случае RBF-сетей, этот параметр выбирается из тех соображений, чтобы «шапки» определенное число раз перекрывались: выбор слишком маленьких отклонений приведет к «острым» аппроксимирующим функциям и неспособности сети к обобщению, а при слишком больших отклонениях будут теряться детали. Требуемое значение несложно найти опытным путем, подбирая его так, чтобы контрольная ошибка была как можно меньше. К счастью, PNN-сети не очень чувствительны к выбору параметра сглаживания.

Наиболее важные *преимущества* PNN-сетей состоят в том, что выходное значение имеет вероятностный смысл (и поэтому его легче интерпретировать), и в том, что сеть быстро обучается. При обучении такой сети время тратится практически только на то, чтобы подавать ей на вход обучающие наблюдения, и сеть работает настолько быстро, насколько это вообще возможно.

Существенным *недостатком* таких сетей является их объем. PNN-сеть фактически вмещает в себя все обучающие данные, поэтому она требует много памяти и может медленно работать.

PNN-сети особенно полезны при пробных экспериментах (например, когда нужно решить, какие из входных переменных использовать), так как благодаря короткому времени обучения можно быстро проделать большое количество пробных тестов.

2.5.6. Обобщенно-регрессионная нейронная сеть (GRNN). Данная сеть устроена аналогично вероятностной нейронной сети (PNN), но она предназначена для решения задач регрессии, а не классификации. Как и в случае PNN-сети, в точку расположения каждого обучающего наблюдения помещается гауссова ядерная функция. Мы считаем, что каждое наблюдение свидетельствует о некоторой нашей уверенности в том, что поверхность отклика в данной точке имеет определенную высоту, и эта уверенность убывает при отходе в сторону от точки. GRNN-сеть копирует внутрь себя все обучающие наблюдения и использует их для оценки отклика в произвольной точке. Окончательная выходная оценка сети получается как взвешенное среднее выходов по всем обучающим наблюдениям:

$$y = \frac{\sum_{k=1}^N y^k \varphi(\|\mathbf{X} - \mathbf{X}^k\|/\sigma)}{\sum_{k=1}^N \varphi(\|\mathbf{X} - \mathbf{X}^k\|/\sigma)},$$

где $\mathbf{X}^k, y^k$ — точки обучающей выборки.

Первый промежуточный слой сети GRNN состоит из радиальных элементов. Второй промежуточный слой содержит элементы, которые помогают оценить взвешенное среднее. Каждый выход имеет в этом слое свой элемент, формирующий для него взвешенную сумму. Чтобы получить из взвешенной суммы взвешенное среднее, эту сумму нужно поделить на сумму весовых коэффициентов. Последнюю сумму вычисляет специальный элемент второго слоя. После этого в выходном слое производится собственно деление (с помощью специальных элементов «деления»). Таким образом, число элементов во втором промежуточном слое на единицу больше, чем в выходном слое. Как правило, в задачах регрессии

требуется оценить одно выходное значение, и, соответственно, второй промежуточный слой содержит два элемента.

Можно модифицировать GRNN-сеть таким образом, чтобы радиальные элементы соответствовали не отдельным обучающим случаям, а их кластерам. Это уменьшает размеры сети и увеличивает скорость обучения. Центры для таких элементов можно выбирать с помощью любого предназначенного для этой цели алгоритма (выборки из выборки, K -средних или Кохонена).

Достоинства и недостатки у сетей GRNN в основном такие же, как и у сетей PNN, единственное различие в том, что GRNN используются в задачах регрессии, а PNN — в задачах классификации. GRNN-сеть обучается почти мгновенно, но может получиться большой и медленной (хотя здесь, в отличие от PNN, не обязательно иметь по одному радиальному элементу на каждый обучающий пример, их число все равно будет большим). Как и сеть RBF, сеть GRNN не обладает способностью экстраполировать данные.

2.5.7. Линейные НС. Согласно общепринятому в науке принципу, если более сложная модель не дает лучших результатов, чем более простая, то из них следует предпочесть вторую. В терминах аппроксимации отображений самой простой моделью будет линейная, в которой аппроксимирующая (подгоночная) функция определяется гиперплоскостью. В задаче классификации гиперплоскость размещается таким образом, чтобы она разделяла собой два класса (линейная дискриминантная функция); в задаче регрессии гиперплоскость должна проходить через заданные точки. Линейная модель обычно задается уравнением

$$Y = XW + B,$$

где W — матрица весов сети, B — вектор смещений.

На языке нейронных сетей линейная модель представляется сетью без промежуточных слоев, которая в выходном слое содержит только линейные элементы (т.е. элементы с линейной функцией активацией). Веса соответствуют элементам матрицы, а пороги — компонентам вектора смещения. Во время работы сеть фактически умножает вектор входов на матрицу весов, а затем к полученному вектору прибавляет вектор смещения.

2.6. Эффективность нейронных сетей

Эффективность нейронных сетей устанавливается рядом так называемых теорем о полноте. Ранее в нестрогой формулировке была приведена одна из них. Рассмотрим еще одну подобную теорему.

В 1989 г. Funahashi показал, что бесконечно большая нейронная сеть с единственным скрытым слоем способна аппроксимировать любую непрерывную функцию, сформулировав данное утверждение в форме следующей теоремы.

Теорема. Пусть $\phi(x)$ — непостоянная, ограниченная и монотонно возрастающая непрерывная функция. Пусть, далее, $U \subset \mathbb{R}^n$ — ограниченное множество и

$$f: U \rightarrow \mathbb{R}$$

вещественная непрерывная функция, определенная на U .

Тогда для произвольного $\varepsilon > 0$ существует целое L и вещественные константы w_i, w_{ij} такие, что аппроксимация

$$\tilde{f}(x_1, x_2, \dots, x_n) = \sum_{i=1}^L w_i \phi \left(\sum_{j=1}^n w_{ij} x_j \right)$$

удовлетворяет неравенству

$$\|f - \tilde{f}\|_{\infty} = \sup_{\mathbf{x} \in U} |f(\mathbf{x}) - \tilde{f}(\mathbf{x})| \leq \varepsilon.$$

Другими словами, любое непрерывное отображение может быть аппроксимировано в смысле однородной топологии на U двухслой-

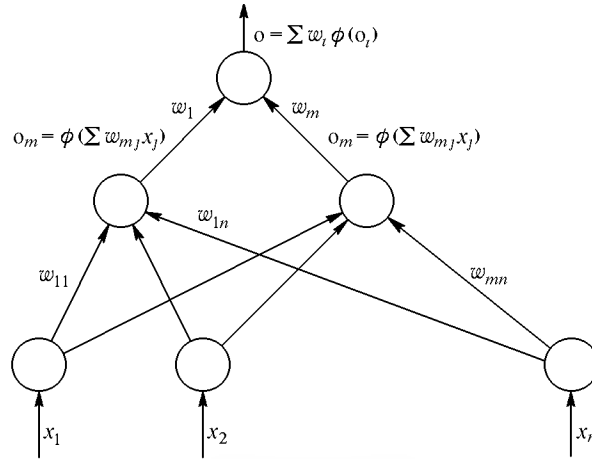


Рис. 2.21. Нейронная сеть Funahashi

ной нейронной сетью с активационными функциями $\phi(x)$ для нейронов скрытого слоя и линейными активационными функциями

для нейронов выходного слоя. На рис. 2.21 представлена НС Funahashi для аппроксимации скалярной функции векторного аргумента.

Отметим еще раз, что приведенная теорема о полноте является далеко не единственной из известных.

Основными недостатками аппарата нейронных сетей являются:

- отсутствие строгой теории по выбору структуры НС;
- практическая невозможность извлечения приобретенных знаний из обученной НС (нейронная сеть практически всегда — «вещь в себе», черный ящик для исследователя).

* * *

Приведенный краткий материал о теории искусственных нейронных сетей является достаточным для перехода к следующей главе. Более подробная информация по структурам, алгоритмам обучения и использованию НС приведена, например, в рекомендуемой литературе.

Глава 3

ГИБРИДНЫЕ СЕТИ

Каждая разновидность систем искусственного интеллекта имеет свои особенности, например, по возможностям обучения, обобщения и выработки выводов, что делает ее наиболее пригодной для решения одного класса задач и менее пригодной — для другого.

Например, нейронные сети хороши для задач распознавания образов, но весьма неудобны для выяснения вопроса, *как они такое распознавание осуществляют*. Они могут автоматически приобретать знания, но процесс их обучения зачастую происходит достаточно медленно, а анализ обученной сети весьма сложен (обученная сеть обычно — черный ящик для пользователя). При этом какую-либо априорную информацию (знания эксперта) для ускорения процесса ее обучения в нейронную сеть ввести невозможно.

Системы с нечеткой логикой, напротив, хороши для объяснения получаемых с их помощью выводов, но *они не могут автоматически приобретать* знания для использования их в механизмах выводов. Необходимость разбиения универсальных множеств на отдельные области, как правило, ограничивает количество входных переменных в таких системах небольшим значением.

Вообще говоря, теоретически, системы с нечеткой логикой и искусственные нейронные сети эквивалентны друг другу, однако, в соответствии с изложенным выше, на практике у них имеются свои собственные достоинства и недостатки. Данное соображение легло в основу аппарата гибридных сетей, в которых выводы делаются на основе аппарата нечеткой логики, но соответствующие функции принадлежности подстраиваются с использованием алгоритма обучения нейронных сетей, например, алгоритма обратного распространения ошибки. Такие системы не только используют априорную информацию, но могут приобретать новые знания и для пользователя являются логически прозрачными.

3.1. Основные понятия и определения гибридных сетей

Для пояснения сущности гибридных сетей, рассмотрим еще раз простую нейронную сеть, имеющую два входа и только один нейрон (рис. 3.1).

Здесь входные сигналы x_i «взаимодействуют» с весами w_i , образуя произведения

$$p_i = x_i w_i, \quad i = 1, 2.$$

Такая частная информация (произведения) объединяются с использованием операции суммирования, образуя вход net нейрона:

$$\text{net} = p_1 + p_2 = w_1 x_1 + w_2 x_2.$$

Выход нейрона образуется в результате преобразования входа net некоторой активационной функцией:

$$y = f(\text{net}) = f(w_1 x_1 + w_2 x_2),$$

например, сигмоидного типа

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Приведенную однострунную сеть, в которой используются операции умножения, суммирования и сигмоидная функция активации, будем называть стандартной нейронной сетью.

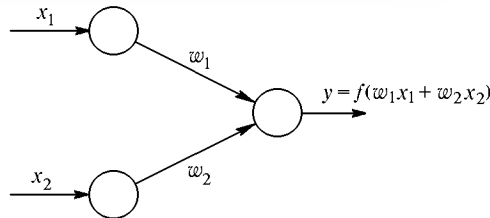


Рис. 3.1. Элементарная НС

В случае применения других операций, таких как t -норма или t -конорма (см. гл. 1), приходим к нейронной сети, которая будет называться *гибридной*.

Определение. Гибридная нейронная сеть — это нейронная сеть с четкими сигналами, весами и активационной функцией, но с объединением x_i и w_i , p_1 и p_2 с использованием t -нормы, t -конормы или некоторых других непрерывных операций.

Входы, выходы и веса гибридной нейронной сети — вещественные числа, принадлежащие отрезку $[0, 1]$.

Рассмотрим следующие примеры элементарных гибридных нейронных сетей.

Нечеткий нейрон «И». Сигналы x_i и веса w_i в данном случае объединяются с помощью треугольной конормы:

$$p_i = S(w_i, x_i), \quad i = 1, 2,$$

а выход образуется с применением треугольной нормы (рис. 3.2):

$$y = \text{AND}(p_1, p_2) = T(p_1, p_2) = T(S(w_1, x_1), S(w_2, x_2)).$$

Если принять

$$T = \min, \quad S = \max,$$

тогда нечеткий нейрон «И» реализует композицию min-max:

$$y = \min(w_1 \vee x_1, w_2 \vee x_2).$$

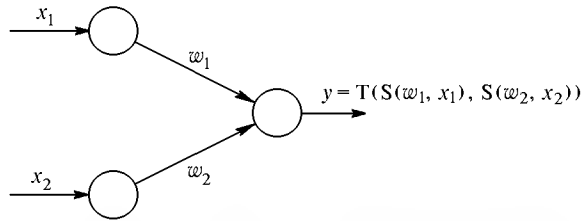


Рис. 3.2. Структура гибридного нейрона «И»

Нечеткий нейрон «ИЛИ». Сигналы x_i и веса w_i здесь объединяются с помощью треугольной нормы:

$$p_i = T(w_i, x_i), \quad i = 1, 2,$$

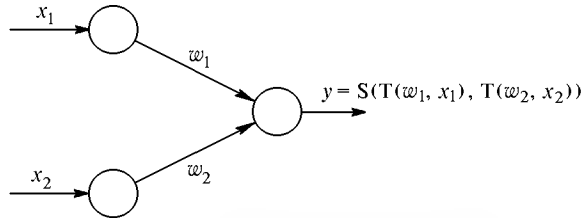


Рис. 3.3. Нечеткий нейрон «ИЛИ»

а выход образуется с применением треугольной конормы (см. рис. 3.3):

$$y = \text{OR}(p_1, p_2) = S(p_1, p_2) = S(T(w_1, x_1), T(w_2, x_2)).$$

Если принять

$$T = \min, \quad S = \max,$$

тогда нечеткий нейрон «ИЛИ» реализует композицию max-min:

$$y = \max(w_1 \wedge x_1, w_2 \wedge x_2).$$

3.2. Алгоритмы обучения и использования гибридных сетей

Опишем типовой подход к построению алгоритмов обучения и использования гибридных нейронных сетей.

Предположим, что гибридной сетью должно быть реализовано (неизвестное) отображение

$$y^k = f(\mathbf{x}^k) = f(x_1^k, x_2^k, \dots, x_n^k), \quad k = 1, 2, \dots, N,$$

при наличии обучающего множества

$$\{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^N, y^N)\}.$$

Для моделирования неизвестного отображения f используем ранее рассмотренный упрощенный алгоритм нечеткого вывода, применяя следующую форму записи предикатных правил:

P_i : если x_1 есть A_{i1} и x_2 есть A_{i2} и ... и x_n есть A_{in} , тогда $y = z_i$, $i = 1, 2, \dots, m$,

где A_{ij} — нечеткие числа треугольной формы, z_i — вещественные числа, определяя степень истинности i -го правила с помощью операции умножения (Larsen):

$$\alpha_i = \prod_{j=1}^n A_{ij}(x_j^k)$$

(здесь можно использовать и другие представления для моделирования логического оператора «И») и определяя выход нечеткой системы дискретным аналогом центроидного метода:

$$o^k = \frac{\sum_{i=1}^m \alpha_i z_i}{\sum_{i=1}^m \alpha_i}.$$

Введение функции ошибки для k -го предъявленного образца вида

$$E_k = \frac{1}{2}(o^k - y^k)^2$$

позволяет, далее, как в обычных (стандартных) нейронных сетях использовать градиентный метод для подстройки параметров заданных предикатных правил. Так, величины z_i можно корректировать по соотношению

$$z_i := z_i - \eta \frac{\partial E_k}{\partial z_i} = z_i - \eta(o^k - y^k) \frac{\alpha_i}{\alpha_1 + \alpha_2 + \dots + \alpha_m},$$

$$i = 1, 2, \dots, m,$$

где η , как и раньше, — константа, характеризующая скорость обучения.

Более детально алгоритм настройки рассмотрим на примере системы, включающей два правила:

П₁: если x есть A_1 , тогда $y = z_1$,

П₂: если x есть A_2 , тогда $y = z_2$,

при этом предполагается, что нечеткие понятия A_1 («малый») и A_2 («большой») имеют сигмоидные функции принадлежности

$$A_1(x) = \frac{1}{1 + e^{b_1(x-a_1)}}, \quad A_2(x) = \frac{1}{1 + e^{b_2(x-a_2)}},$$

характеризующиеся параметрами a_1, a_2, b_1, b_2 .

Степени истинности правил определяются в данном случае соотношениями

$$\alpha_1 = A_1(x) = \frac{1}{1 + e^{b_1(x-a_1)}}, \quad \alpha_2 = A_2(x) = \frac{1}{1 + e^{b_2(x-a_2)}},$$

а выход системы — выражением

$$o = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2} = \frac{A_1(x) z_1 + A_2(x) z_2}{A_1(x) + A_2(x)}.$$

Предположим, что имеется обучающее множество $\{(x_1, y_1), \dots, (x^N, y^N)\}$, отображающее неизвестную функцию f .

Требуется: осуществить такую настройку параметров системы $a_1, a_2, b_1, b_2, z_1, z_2$, при которой обеспечивается наилучшая аппроксимация данной функции.

Решение. В данном случае функция ошибки может быть записана в форме

$$E_k = E_k(a_1, b_1, a_2, b_2, z_1, z_2) = \frac{1}{2} (o^k(a_1, b_1, a_2, b_2, z_1, z_2) - y^k)^2.$$

Используя далее тот же подход, что и при выводе алгоритма обратного распространения ошибки (см. § 2.5), запишем:

$$\begin{aligned}
 z_1 &:= z_1 - \eta \frac{\partial E_k}{\partial z_1} = z_1 - \eta(o^k - y^k) \frac{\alpha_1}{\alpha_1 + \alpha_2} = \\
 &= z_1 - \eta(o^k - y^k) \frac{A_1(x^k)}{A_1(x^k) + A_2(x^k)}, \\
 z_2 &:= z_2 - \eta \frac{\partial E_k}{\partial z_2} = z_2 - \eta(o^k - y^k) \frac{\alpha_2}{\alpha_1 + \alpha_2} = \\
 &= z_2 - \eta(o^k - y^k) \frac{A_2(x^k)}{A_1(x^k) + A_2(x^k)}.
 \end{aligned}$$

Аналогичным путем могут быть получены развернутые выражения для коррекции коэффициентов a_1 , a_2 , b_1 , b_2 . Исходные

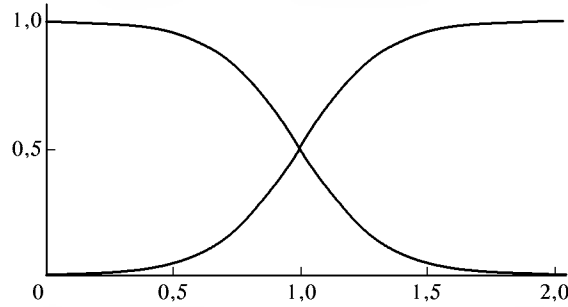


Рис. 3.4. Симметричные функции принадлежности

соотношения таковы:

$$\begin{aligned}
 a_1 &:= a_1 - \eta \frac{\partial E_k}{\partial a_1}, & a_2 &:= a_2 - \eta \frac{\partial E_k}{\partial a_2}, \\
 b_1 &:= b_1 - \eta \frac{\partial E_k}{\partial b_1}, & b_2 &:= b_2 - \eta \frac{\partial E_k}{\partial b_2}.
 \end{aligned}$$

Конечные выражения являются достаточно громоздкими, но могут быть упрощены в случае, если функции принадлежности имеют вид

$$A_1(x) = \frac{1}{1 + e^{-b(x-a)}}, \quad A_2(x) = \frac{1}{1 + e^{b(x-a)}}.$$

Данные функции характеризуются всего двумя параметрами (a и b), в определенном смысле являются симметричными (см. рис. 3.4) и удовлетворяют уравнению

$$A_1(x) + A_2(x) = 1.$$

Заметим, что из последнего и ранее полученных уравнений следует:

$$\begin{aligned} z_1 &:= z_1 - \eta \frac{\partial E_k}{\partial z_1} = z_1 - \eta(o^k - y^k) \frac{\alpha_1}{\alpha_1 + \alpha_2} = \\ &= z_1 - \eta(o^k - y^k) A_1(x^k), \end{aligned}$$

$$\begin{aligned} z_2 &:= z_2 - \eta \frac{\partial E_k}{\partial z_2} = z_2 - \eta(o^k - y^k) \frac{\alpha_2}{\alpha_1 + \alpha_2} = \\ &= z_2 - \eta(o^k - y^k) A_2(x^k). \end{aligned}$$

Последующие выкладки таковы:

$$a := a - \eta \frac{\partial E_k(a, b)}{\partial a},$$

где

$$\begin{aligned} \frac{\partial E_k(a, b)}{\partial a} &= (o^k - y^k) \frac{\partial o^k}{\partial a} = \\ &= (o^k - y^k) \frac{\partial}{\partial a} (z_1 A_1(x^k) + z_2 A_2(x^k)) = \\ &= (o^k - y^k) \frac{\partial}{\partial a} (z_1 A_1(x^k) + z_2 (1 - A_1(x^k))) = \\ &= (o^k - y^k) (z_1 - z_2) \frac{\partial A_1(x^k)}{\partial a} = (o^k - y^k) (z_1 - z_2) b \times \\ &\times \frac{e^{b(x^k - a)}}{(1 + e^{b(x^k - a)})^2} = (o^k - y^k) (z_1 - z_2) b A_1(x^k) (1 - A_1(x^k)) = \\ &= (o^k - y^k) (z_1 - z_2) b A_1(x^k) A_1(x^k), \end{aligned}$$

и

$$b := b - \eta \frac{\partial E_k(a, b)}{\partial b},$$

где

$$\begin{aligned}\frac{\partial E_k(a, b)}{\partial b} &= (o^k - y^k)(z_1 - z_2) \frac{\partial}{\partial b} \frac{1}{1 + e^{b(x^k - a)}} = \\ &= (o^k - y^k)(z_1 - z_2)(x^k - a)A_1(x^k)(1 - A_1(x^k)) = \\ &= (o^k - y^k)(z_1 - z_2)(x^k - a)A_1(x^k)A_1(x^k).\end{aligned}$$

Приведенные выкладки, как представляется, полностью иллюстрируют идеи алгоритмов обучения и использования гибридной сети.

Другим примером может служить система, имеющая следующую базу знаний:

П₁: если x_1 есть L₁ и x_2 есть L₂ и x_3 есть L₃, тогда y есть H,

П₂: если x_1 есть H₁ и x_2 есть H₂ и x_3 есть L₃, тогда y есть M,

П₃: если x_1 есть H₁ и x_2 есть H₂ и x_3 есть H₃, тогда y есть S,

где x_1, x_2, x_3 — входные переменные, y — выход системы, L₁, L₂, L₃, H₁, H₂, H₃, H, M, S — некоторые нечеткие множества с функциями принадлежности сигмоидного типа:

$$\begin{aligned}L_j(t) &= \frac{1}{1 + e^{b_j(t - c_j)}}, \quad H_j(t) = \frac{1}{1 + e^{-b_j(t - c_j)}}, \quad j = 1, 2, 3, \\ H(t) &= \frac{1}{1 + e^{-b_4(t - c_4 + c_5)}}, \quad M(t) = \frac{1}{1 + e^{-b_4(t - c_4)}}, \\ S(t) &= \frac{1}{1 + e^{b_4(t - c_4)}}.\end{aligned}$$

Для определения выходной переменной используется алгоритм вывода Tsukamoto (см. выше), т.е.

1) подсчитываются значения истинности предпосылок для каждого правила:

$$\alpha_1 = L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3),$$

$$\alpha_2 = H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3),$$

$$\alpha_3 = H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3),$$

где в данном случае a_1, a_2, a_3 — текущие значения входов системы;

2) для каждого правила определяются частные выходы:

$$z_1 = B^{-1}(\alpha_1) = c_4 + c_5 + \frac{1}{b_4} \ln \frac{1 - \alpha_1}{\alpha_1},$$

$$z_2 = B^{-1}(\alpha_2) = c_4 + \frac{1}{b_4} \ln \frac{1 - \alpha_2}{\alpha_2},$$

$$z_3 = B^{-1}(\alpha_3) = c_4 + \frac{1}{b_4} \ln \frac{1 - \alpha_3}{\alpha_3};$$

3) находится общий выход системы:

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3}.$$

Изложенный процесс иллюстрируется рис. 3.5.

Гибридная нейронная сеть, отражающая приведенный механизм вывода, представлена на рис. 3.6. Заметим, что сети с подоб-

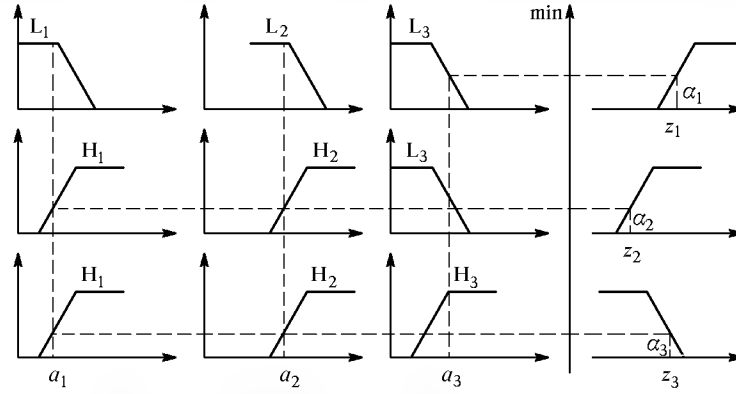


Рис. 3.5. Иллюстрация алгоритма вывода Tsukamoto

ной архитектурой в англоязычной литературе получили название ANFIS (Adaptive Neuro-Fuzzy Inference System).

Данная сеть может быть описана следующим образом.

1. *Слой 1 (Layer 1)*. Выходы узлов этого слоя представляют собой значения функций принадлежности при конкретных (заданных) значениях входов.

2. *Слой 2 (Layer 2)*. Выходами нейронов этого слоя являются степени истинности предпосылок каждого правила базы знаний системы, вычисляемые по формулам:

$$\alpha_1 = L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3),$$

$$\alpha_2 = H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3),$$

$$\alpha_3 = H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3).$$

Все нейроны этого слоя обозначены буквой Т, что означает, что они могут реализовывать произвольную t -норму для моделирования операции «И».

3. *Слой 3 (Layer 3)*. Нейроны этого слоя (обозначены буквой N) вычисляют величины:

$$\beta_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}, \quad \beta_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2 + \alpha_3}, \quad \beta_3 = \frac{\alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}.$$

4. *Слой 4 (Layer 4)*. Нейроны данного слоя выполняют операции:

$$\beta_1 z_1 = \beta_1 H^{-1}(a_1), \quad \beta_2 z_2 = \beta_2 M^{-1}(a_2), \quad \beta_3 z_3 = \beta_3 S^{-1}(a_3).$$

5. *Слой 5 (Layer 5)*. Единственный нейрон этого слоя вычисляет выход сети:

$$z_0 = \beta_1 z_1 + \beta_2 z_2 + \beta_3 z_3.$$

Корректировка параметров системы здесь производится в соответствии с ранее рассмотренным подходом. Так, например, на-

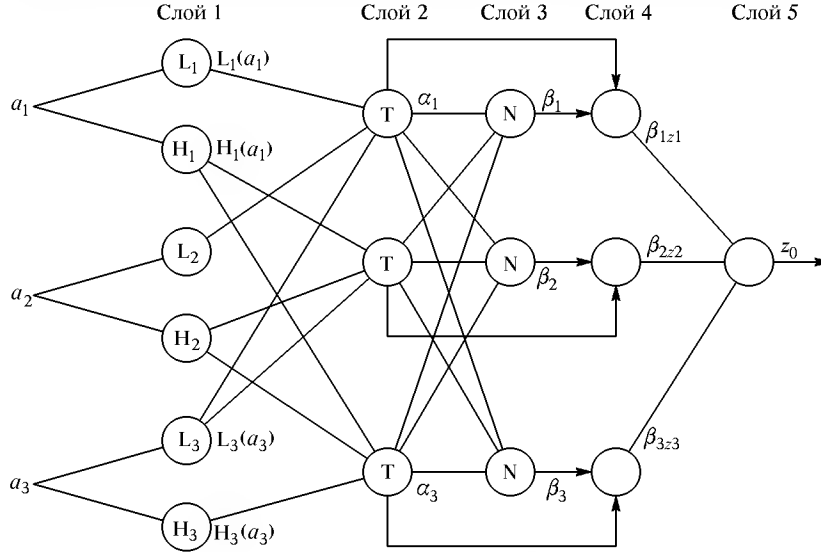


Рис. 3.6. Структура гибридной нейронной сети (архитектура ANFIS)

стройка коэффициентов b_4 , c_4 и c_5 по формулам:

$$b_4 := b_4 - \eta \frac{\partial E_k}{\partial b_4} = b_4 - \frac{\eta}{b_4^2} \delta_k \frac{a_1 + a_2 - a_3}{a_1 + a_2 + a_3},$$

$$c_4 := c_4 - \eta \frac{\partial E_k}{\partial c_4} = c_4 + \eta \delta_k \frac{a_1 + a_2 + a_3}{a_1 + a_2 + a_3} = c_4 + \eta \delta_k,$$

$$c_5 := c_5 - \eta \frac{\partial E_k}{\partial c_5} = c_5 + \eta \delta_k \frac{a_1}{a_1 + a_2 + a_3},$$

где $\delta_k = y^k - o^k$.

Соответствующие выражения могут быть получены и для остальных коэффициентов.

3.3. Нечеткий гибридный классификатор

Рассмотрим теперь, как с помощью гибридной системы решается задача классификации, т.е. отнесение объекта, характеризующегося набором признаков, к некоторому классу.

Одна из возможных структур для решения подобной задачи приведена на рис. 3.7.

Предполагается, что здесь объект характеризуется двумя количественными признаками x_1 и x_2 и относится к одному из двух

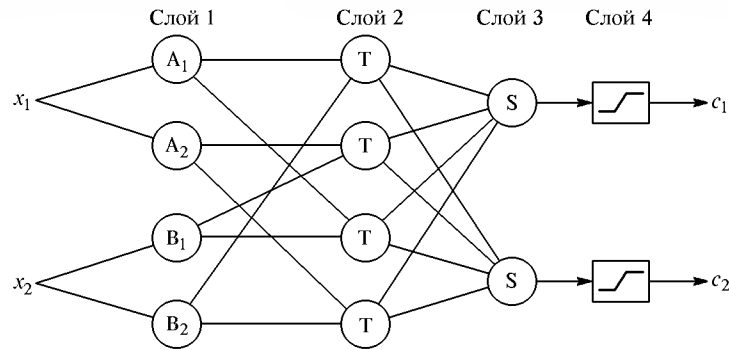


Рис. 3.7. Гибридная сеть для решения задачи классификации

классов — c_1 или c_2 . Каждый вход представляется двумя лингвистическими понятиями, что позволяет ограничиться всего четырьмя правилами.

Сеть может быть описана следующим образом.

1. *Слой 1 (Layer 1)*. Выходы узлов данного слоя — это степени принадлежности входных переменных определенным для них нечетким множествам A_1, A_2, B_1, B_2 .

В данном случае выбраны функции принадлежности колоколообразного вида

$$A_i(t) = \exp(-(t - a_{i1}/b_{i1})^2/2)$$

с набором параметров $a_{i1}, a_{i2}, b_{i1}, b_{i2}$.

Значения данных параметров корректируются в процессе обучения сети (основанном на градиентном методе).

2. *Слой 2 (Layer 2)*. Каждый нейрон этого слоя является нейроном типа рассмотренного выше гибридного (нечеткого) нейрона «И».

3. *Слой 3 (Layer 3)*. Нейроны данного слоя являются обычными (стандартными) нейронами, входами которых являются линейные (взвешенные) комбинации выходов нейронов предыдущего

слоя, а выходы формируются с использованием активационных функций сигмоидного типа. Эти выходы трактуются как степени принадлежности предъявленного объекта первому или второму классу.

Алгоритм обучения данной сети, в принципе, не отличается от рассмотренных.

3.4. Программная реализация моделей нечеткой логики, нейросетевых и гибридных

В настоящее время существует огромное количество программных продуктов, позволяющих реализовывать нейросетевые структуры (так называемых программ-нейроимитаторов). Значительно меньше программ, работающих с моделями нечеткой логики (и они пока практически недоступны отечественным специалистам). Отметим в этой связи, что для целей проектирования и использования как нейросетевых моделей, так и нечетких и гибридных крайне удобной является достаточно распространенная в нашей стране математическая система MATLAB (версии 5.2 и 5.3), точнее два инструментальных средства этой системы: пакеты Neural Networks Toolbox (нейронные сети) и Fuzzy Logic Toolbox (пакет нечеткой логики), которые и будут ниже подробно рассмотрены.

Глава 4

ПАКЕТ NEURAL NETWORKS TOOLBOX

4.1. Назначение пакета Neural Networks Toolbox

Пакет Neural Networks Toolbox (нейронные сети) содержит средства для проектирования, моделирования, обучения и использования множества известных парадигм аппарата искусственных нейронных сетей (ИНС), от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для решения множества разнообразных задач, таких как обработка сигналов, нелинейное управление, финансовое моделирование и т. п.

Для каждого типа архитектуры и обучающего алгоритма ИНС имеются функции инициализации, обучения, адаптации, создания и моделирования, демонстрации и примеры применения.

4.2. Обзор функций пакета Neural Networks Toolbox

В состав пакета Neural Networks входят более 150 различных функций, образуя собой своеобразный макроязык программирования и позволяя пользователю создавать, обучать и использовать самые различные НС. С помощью команды

» **help nnet**

можно получить перечень входящих в пакет функций. Для получения справки по любой функции можно использовать команду

» **help имя_функции**

Данные функции по своему назначению делятся на ряд групп. Рассмотрим основные из них.

4.2.1. Функции активации (передаточные функции) и связанные с ними функции

compet(X) — функция конкуренции — в качестве аргумента использует матрицу **X**, столбцы которой ассоциируются с векторами входов. Возвращает разреженную матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца.

Пример

```
» X = [0.9 -0.6; 0.1 0.4; 0.2 -0.5; 0 0.5];
» compet(X)
ans =
(1,1)    1
(4,2)    1
```

В форме **compet(code)**, где переменная **code** может принимать значения **'deriv'** (имя производной функции), **'name'** (полное имя), **'output'** (диапазон выхода), **'active'** (возможный диапазон входов).

Примеры

```
» compet('deriv')
ans =
''
» compet('name')
ans =
Competitive
» compet('output')
ans =
0      1
» compet('active')
ans =
-Inf   Inf
```

Данная функция используется при создании НС со слоем «состязующихся» нейронов (как, например, в сетях встречного распространения).

hardlim(X) пороговая функция активации с порогом $\theta = 0$; аргумент имеет тот же смысл, что и для предыдущей команды. Возвращает матрицу, размер которой равен размеру матрицы **X**, а элементы имеют значения 0 или 1 в зависимости от знака соответствующего элемента **X**.

Пример

```
» X = [0.9 -0.6; 0.1 0.4; 0.2 -0.5; 0 0.5];
» hardlim(X)
ans =
1      0
1      1
1      0
1      1
```

В форме **hardlim(code)** возвращает информацию, аналогичную рассмотренной для команды **compet**.

hardlims(X) — знаковая или сигнатурная функция активации; действует так же, как функция **hardlim(X)**, но возвращает значения -1 или $+1$.

logsig(X) — сигмоидальная логистическая функция. Возвращает матрицу, элементы которой являются значениями логистической функции (см. табл. 2.1) от аргументов — элементов матрицы **X**.

poslin(X) — возвращает матрицу значений полулинейной функции (табл. 2.1).

purelin(X) — возвращает матрицу значений линейной функции активации (табл. 2.1).

radbas(X) — возвращает матрицу значений радиальной базисной функции (табл. 2.1).

satlin(X) — возвращает матрицу значений полулинейной функции с насыщением (табл. 2.1).

satlins(X) — возвращает матрицу значений линейной функции с насыщением (табл. 2.1).

softmax(X) — возвращает матрицу, элементы которой вычисляются по формуле

$$\frac{\exp(x_{ij})}{\sum_{i=1}^N \exp(x_{ij})}$$

где N — число строк матрицы-аргумента **X**.

tansig(X) — возвращает матрицу значений сигмоидальной (гиперболический тангенс) функции (см. табл. 2.1).

tribas(X) — возвращает матрицу значений треугольной функции принадлежности (см. табл. 2.1).

dhardlim(X,Y) — производная пороговой функции активации. Аргументами являются матрица входов **X** и матрица выходов **Y**; матрицы имеют одинаковый размер. Возвращается матрица того же размера с нулевыми элементами.

dhardlms(X,Y) — производная знаковой функции активации (см. табл. 2.1). Возвращается матрица с нулевыми элементами.

dlogsig(X,Y) — производная сигмоидальной логистической функции. Возвращается матрица с элементами $y_{ij}(1 - y_{ij})$.

П р и м е р ы

» **X** = [0.1; 0.8; -0.7];

» **Y** = **logsig(X)**

Y =

0.5250

```

0.6900
0.3318
» dY_dX = dlogsig(X,Y)
dY_dX =
0.2494
0.2139
0.2217

```

dposlin(X,Y) — производная полулинейной функции. Возвращается матрица с элементами, равными единице для соответственных положительных элементов матрицы-аргумента **Y** и равными нулю в противоположном случае.

dpurelin(X,Y) — производная линейной функции активации. Возвращается матрица с единичными элементами.

dradbas(X,Y) — производная радиальной базисной функции (см. табл. 2.1). Возвращается матрица с элементами $-2x_{ij}y_{ij}$.

dsatlin(X,Y) — возвращает матрицу значений производной полулинейной функции с насыщением (табл. 2.1). Элементы такой матрицы — единицы, если соответственные элементы матрицы **Y** принадлежат интервалу $(0, 1)$, и нули в противоположном случае.

dsatlins(X,Y) — возвращает матрицу значений производной линейной функции с насыщением (табл. 2.1). Элементы такой матрицы — единицы, если соответственные элементы матрицы **Y** принадлежат интервалу $(-1, 1)$, и нули в противоположном случае.

dtansig(X,Y) — возвращает матрицу значений производной сигмоидальной функции — гиперболического тангенса (табл. 2.1). Элементы этой матрицы определяются выражением $1 - y_{ij}^2$.

dttribas(X,Y) — возвращает матрицу значений производной треугольной функции активации (табл. 2.1). Элементы этой матрицы определяются выражением: 1, если $-1 < y_{ij} < 0$; -1 , если $0 \leq y_{ij} < 1$; 0 в противоположном случае.

4.2.2. Функции обучения нейронных сетей. Позволяют устанавливать алгоритм и параметры обучения НС заданной конфигурации по желанию пользователя. В группу входят следующие функции.

[net,tr] = trainbfg(net,Pd,Tl,Ai,Q,TS,VV,TV) — функция обучения, реализующая разновидность квазиньютоновского алгоритма обратного распространения ошибки (BFGS). Аргументы функции:

net — имя обучаемой НС,
Pd — наименование массива задержанных входов обучающей выборки,
TI — массив целевых значений выходов,

Ai — матрица начальных условий входных задержек,
Q — количество обучающих пар в одном цикле обучения (размер «пачки»),
TS — вектор временных интервалов,
VV — пустой ([]) массив или массив проверочных данных,
TV — пустой ([]) массив или массив тестовых данных.

Функция возвращает обученную нейронную сеть **net** и набор записей **tr** для каждого цикла обучения (**tr.epoch** — номер цикла, **tr.perf** — текущая ошибка обучения, **tr.vperf** — текущая ошибка для проверочной выборки, **tr.tperf** — текущая ошибка для тестовой выборки).

Процесс обучения происходит в соответствии со значениями следующих параметров (в скобках приведены значения по умолчанию):

net.trainParam.epochs (100) — заданное количество циклов обучения,
net.trainParam.show (25) — количество циклов для показа промежуточных результатов,
net.trainParam.goal (0) — целевая ошибка обучения,
net.trainParam.time (∞) — максимальное время обучения в секундах,
net.trainParam.min_grad (10^{-6}) — целевое значение градиента,
net.trainParam.max_fail (5) — максимально допустимая кратность превышения ошибки проверочной выборки по сравнению с достигнутым минимальным значением,
net.trainParam.searchFcn ('srchcha') — имя используемого одномерного алгоритма оптимизации.

Структуры и размеры массивов:

Pd $N_o \times N_i \times TS$ — клеточный массив, каждый элемент которого **P{i,j,ts}** есть матрица $D_{ij} \times Q$,

TI $NI \times TS$ — клеточный массив, каждый элемент которого **P{i,ts}** есть матрица $V_i \times Q$,

Ai $NI \times LD$ — клеточный массив, каждый элемент которого **Ai{i,k}** есть матрица $S_i \times Q$,

где

N_i = **net.numInputs** (количество входов сети),

NI = **net.numLayers** (количество ее слоев),

LD = **net.numLayerDelays** (количество слоев задержки),

R_i = **net.inputs{i}.size** (размер i -го входа),

S_i = **net.layers{i}.size** (размер i -го слоя),

V_i = **net.targets{i}.size** (размер целевого вектора),

D_{ij} = $R_i * \text{length}(\text{net.inputWeights{i,j}.delays})$.

Если массив **VV** не пустой, то он должен иметь структуру, определяемую следующими компонентами:

VV.PD — задержанные значения входов проверочной выборки,
VV.TI — целевые значения,
VV.Ai — начальные входные условия,
VV.Q — количество проверочных пар в одном цикле обучения,
VV.TS — временные интервалы проверочной выборки.

Эти параметры используются для задания останова процесса обучения в случаях, когда ошибка для проверочной выборки не уменьшается или начинает возрастать.

Структуру, аналогичную структуре массива **VV**, имеет массив **TV**.

Рассматриваемая функция, заданная в форме **trainbfg(code)**, возвращает для значений аргумента, соответственно, '**pnames**' и '**pdefaults**', имена параметров обучения и их значения по умолчанию.

Для использования функции в НС, определяемых пользователем, необходимо:

- 1) установить параметр **net.trainFcn = 'trainbfg'** (при этом параметры алгоритма будут заданы по умолчанию);
- 2) установить требуемые значения параметров (**net.trainParam**).

Процесс обучения останавливается в случае выполнения любого из следующих условий:

- превышено заданное количество циклов обучения (**net.trainParam.epochs**),
- превышено заданное время обучения (**net.trainParam.time**),
- ошибка обучения стала меньше заданной (**net.trainParam.goal**),
- градиент стал меньше заданного (**net.trainParam.min_grad**),
- возрастание ошибки проверочной выборки по сравнению с достигнутым минимальным превысило заданное значение (**net.trainParam.max_fail**).

Пример

```
» P = [0 1 2 3 4 5]; % Задание входного вектора
» T = [0 0 0 1 1 1]; % Задание целевого вектора
» % Создание и тестирование сети
» net = newff([0 5],[2 1],{'tansig','logsig'},'traincgf');
» a = sim(net,P)
a =
    0.0586    0.0772    0.0822    0.0870    0.1326    0.5901
» % Обучение с новыми параметрами и повторное тестирование
```

```

» net.trainParam.searchFcn = 'srchcha';
» net.trainParam.epochs = 50;
» net.trainParam.show = 10;
» net.trainParam.goal = 0.1;
» net = train(net,P,T);
TRAINCGF-srchcha, Epoch 0/50, MSE 0.295008/0.1,
Gradient 0.623241/1e-006
TRAINCGF-srchcha, Epoch 1/50, MSE 0.00824365/0.1,
Gradient 0.0173555/1e-006
TRAINCGF, Performance goal met.
» a = sim(net,P)
a =
    0.0706    0.1024    0.1474    0.9009    0.9647    0.9655

```

В данном примере созданная многослойная НС, обученная с установками по умолчанию, вначале показала плохой результат отображения обучающей выборки, но после изменения параметров и повторного обучения сети результат стал вполне приемлемым.

[net,tr] = trainbr(net,Pd,TL,Ai,Q,TS,VV) — функция, реализующая так называемый Байесовский метод обучения, сущность которого заключается в подстройке весов и смещений сети на основе алгоритма Левенберга-Марквардта. Данный алгоритм минимизирует комбинацию квадратов ошибок и весов с выбором наилучшей такой (для получения наилучших обобщающих свойств сети). Эта процедура известна как Байесовская регуляризация, откуда следует название метода.

Аргументы, параметры, возвращаемые величины и использование — такие же, как у предыдущей функции. Сказанное остается в силе для всех остальных функций данной группы.

[net,tr] = traincgb(net,Pd,TL,Ai,Q,TS,VV) — функция обучения НС, реализующая разновидность алгоритма сопряженных градиентов (так называемый метод Powell Beale).

[net,tr] = traincgf(net,Pd,TL,Ai,Q,TS,VV) — функция обучения НС, реализующая разновидность алгоритма обратного распространения ошибки в сочетании с методом оптимизации Флетчера-Поуэлла.

[net,tr] = traincgp(net,Pd,TL,Ai,Q,TS,VV) — то же, что в предыдущем случае, но с использованием метода Polak-Ribiere.

[net,tr] = traingd(net,Pd,TL,Ai,Q,TS,VV) — функция, реализующая «классический» алгоритм обратного распространения ошибки.

[net,tr] = traingda(net,Pd,TL,Ai,Q,TS,VV) — то же, что в предыдущем случае, но с адаптацией коэффициента скорости обучения.

[net,tr] = traingdm(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая модифицированный алгоритм обратного распространения ошибки с введенной «инерционностью» коррекции весов и смещений.

[net,tr] = traingdx(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая комбинированный алгоритм обучения, объединяющий особенности двух вышеприведенных.

[net,tr] = trainlm(net,Pd,TI,Ai,Q,TS,VV) — данная функция возвращает веса и смещения НС, используя алгоритм оптимизации Левенберга-Марквардта.

[net,tr] = trainoss(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая разновидность алгоритма обратного распространения ошибки с использованием метода секущих.

[net,tr] = trainrp(net,Pd,TI,Ai,Q,TS,VV) — функция, реализующая разновидность алгоритма обратного распространения ошибки, так называемый упругий алгоритм обратного распространения (resilient backpropagation algorithm, RPROP).

[net,tr] = trainscg(net,Pd,TI,Ai,Q,TS,VV) — данная функция возвращает веса и смещения НС, используя алгоритм масштабируемых сопряженных градиентов.

[net,tr] = trainwb(net,Pd,TI,Ai,Q,TS,VV) — данная функция корректирует веса и смещения сети в соответствии с заданной функцией обучения нейронов.

[net,tr] = trainwb1(net,Pd,TI,Ai,Q,TS,VV) — то же, что и предыдущая функция, но одновременно на вход сети предъявляется только один вектор входа.

[net,Ac,EI] = adaptwb(net,Pd,TI,Ai,Q,TS) — функция адаптации весов и смещений НС. Используется совместно с функциями (см. ниже) **newp** и **newlin**. Возвращает массив выходов слоев **Ac** и массив ошибок слоев **EI**.

4.2.3. Функции настройки слоев нейронов. Функции данной группы являются вспомогательными при работе с некоторыми рассмотренными функциями обучения НС (например, **trainwb**, **trainwb1**, **adaptwb**), а также используются при настройках однослойных нейросетевых структур (персептронов, слоев Кохонена и т. п.).

[dB,LS] = learncon(B,P,Z,N,A,T,E,gW,gA,D,LP,LS) — функция настройки весов с введением «чувства справедливости» (см. выше). Аргументы:

B — $S \times 1$ вектор смещений,

P — $1 \times Q$ входной вектор,

Z — $S \times Q$ матрица взвешенных входов,

N $S \times Q$ матрица входов,
A $S \times Q$ матрица выходных векторов,
T $S \times Q$ матрица целевых векторов слоя,
E $S \times Q$ матрица ошибок,
gW $S \times R$ градиент критерия эффективности по отношению к вектору весов,
gA $S \times Q$ градиент критерия эффективности по отношению к вектору выхода,
D $S \times S$ матрица расстояний между нейронами,
LP — параметр обучения, $LP = []$,
LS — состояние обучения, в начале $[]$.
 Возвращаемые величины:
dB $S \times 1$ вектор изменений весов (или смещений),
LS — новое состояние обучения.

Функция в форме **learncon(code)** возвращает следующую информацию:

при аргументе **'pnames'** — имена параметров обучения,
 при **'pdefaults'** — значения параметров по умолчанию,
 при **'needg'** — 1, если эта функция использует **gW** или **gA**.

Алгоритм выполнения функции сначала вычисляет «чувство справедливости» нейрона по выражению $\mathbf{c} = (1 - \mathbf{lr}) * \mathbf{c} + \mathbf{lr} * \mathbf{a}$, а уже затем корректирует вес в соответствии с формулой

$$\mathbf{b} = \exp(1 - \log(\mathbf{c})) - \mathbf{b}.$$

Пример

```

» a = rand(3,1);
» b = rand(3,1);
» lp.lr = 0.5; % Задание параметра обучения
» dW = learncon(b,[],[],[],a,[],[],[],[],lp,[])
dW =
    0.3449
    0.7657
    0.5405
  
```

learngd — функция коррекции весов и смещений, реализующая градиентный алгоритм оптимизации.

Запись:

```

[dW,LS] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learngd(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learngd(code)
  
```

Описание. Аргументы функции: **W** — матрица весов или вектор смещения, остальные аргументы — как у предыдущей функции.

Возвращаемые параметры — как у предыдущей функции.

learnngdm функция практически аналогична предыдущей, но используемый алгоритм оптимизации — градиентный метод с инерционной составляющей.

$[dW, LS] = \text{learnh}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ — функция коррекции весов, использующая правило Хебба, в соответствии с которым веса корректируются по выражению $dw = lr * a * p'$.

$[dW, LS] = \text{learnhd}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ функция реализует модификацию правила Хебба, при котором корректировка весов осуществляется по соотношению: $dw = lr * a * p' - dr * w$.

$[dW, LS] = \text{learnis}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ функция подстройки весов «входной звезды» (нейрона слоя Гроссберга), реализующая выражение: $dw = lr * a * (p' - w)$.

$[dW, LS] = \text{learnk}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ функция подстройки весов слоя Кохонена, реализующая выражение $dw = lr * (p' - w)$, если $a \neq 0$, и 0 в противном случае.

$[dW, LS] = \text{learnlv1}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ и

$[dW, LS] = \text{learnlv2}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ — функции настройки сетей встречного распространения.

$[dW, LS] = \text{learnos}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ — функция настройки нейрона типа «выходная звезда», реализующая выражение: $dw = lr * (a - w) * p'$.

$[dW, LS] = \text{learnp}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ функция, реализующая алгоритм обучения персептрона.

$[dW, LS] = \text{learnpn}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ — то же, что и предыдущая функция, но с нормализацией входов. Более эффективна при больших изменениях входных сигналов.

$[dW, LS] = \text{learnsom}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ — функция обучения самоорганизующихся карт.

$[dW, LS] = \text{learnwh}(W, P, Z, N, A, T, E, gW, gA, D, LP, LS)$ функция обучения, реализующая так называемый алгоритм Видрова-Хоффа (Widrow-Hoff), основанный на соотношении

$dw = lr * e * pn'$ и известный также, как дельта-правило или правило наименьших квадратов.

4.2.4. Функции одномерной оптимизации. Функции данной группы можно рассматривать как вспомогательные для функций обучения нейронных сетей. Реализуют различные алгоритмы одномерного поиска.

srchbac функция реализует так называемый алгоритм перебора с возвратами (backtracking).

srchbre функция реализует комбинированный метод оптимизации, объединяющий метод золотого сечения и квадратичную интерполяцию.

srchcha — функция реализует разновидность метода оптимизации с применением кубической интерполяции.

srchgol — функция реализует метод золотого сечения.

srchhyb — функция реализует комбинированный метод оптимизации, объединяющий метод дихотомии и кубическую интерполяцию.

4.2.5. Функции инициализации слоев и смещений. Для многих нейронных сетей этапом, предваряющим процедуру их обучения, является этап инициализации (задания некоторых — обычно выбираемых случайным образом) весов и смещений сети. Такая инициализация выполняется с помощью функций данной группы.

initcon(S,PR) — функция, устанавливающая смещения нейронов в зависимости от среднего выхода нейрона. Аргументы: **s** — количество нейронов, **PR** = [**Pmin Pmax**] — матрица (с двумя столбцами) минимальных и максимальных значений входов, по умолчанию [1 1]. Возвращает вектор смещений. Используется совместно с командой **learncon**.

Пример

```
» b = initcon(3)
```

```
b =  
 8.1548  
 8.1548  
 8.1548
```

initzero — функция задания нулевых начальных значений весам или смещениям. Аргументы те же, что и у предыдущей команды.

midpoint(S,PR) — функция инициализации, устанавливающая веса в соответствии со средними значениями входов.

randnc(S,R) — функция задания матрицы весов. Возвращает матрицу размера **S**×**R** со случайными элементами, нормализованную по столбцам (векторы-столбцы имеют единичную длину).

randnr(S,R) — то же, что предыдущая функция, но возвращает матрицу весов, нормализованную по строкам.

rands — функция инициализации весов/смещений заданием их случайных значений из диапазона [−1, 1].

Запись:

```
W = rands(S,PR)
```

```
M = rands(S,R)
```

```
v = rands(S)
```

Описание. Аргументы те же, что и для функции **initcon**; значение **R** по умолчанию — 1. Возвращается матрица соответствующего размера.

4.2.6. Функции создания нейронных сетей

network — функция создания нейронной сети пользователя.

Запись:

net = network

net =

**network(numInputs,numLayers,biasConnect,inputConnect,
layerConnect,outputConnect,targetConnect)**

Описание. Функция возвращает созданную нейронную сеть с именем **net** и со следующими характеристиками (в скобках даны значения по умолчанию):

numInputs — количество входов (0).

numLayers — количество слоев (0),

biasConnect — булевский вектор с числом элементов, равным количеству слоев (нули),

inputConnect — булевская матрица с числом строк, равным количеству слоев, и числом столбцов, равным количеству входов (нули),

layerConnect — булевская матрица с числом строк и столбцов, равным количеству слоев (нули),

outputConnect — булевский вектор-строка с числом элементов, равным количеству слоев (нули).

targetConnect — вектор-строка, такая же, как предыдущая (нули).

net = newc(PR,S,KLR,CLR) — функция создания слоя Кохонена. Функция использует аргументы:

PR — $R \times 2$ матрицу минимальных и максимальных значений для R входных элементов,

S — число нейронов,

KLR — коэффициент обучения Кохонена (по умолчанию 0.01),

CLR — коэффициент «справедливости» (по умолчанию 0.001) и возвращает слой Кохонена с заданным именем.

net = newcf(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF)

функция создания разновидности многослойной НС с обратным распространением ошибки — так называемой каскадной НС. Такая сеть содержит скрытых NI слоев, использует входные функции типа **dotprod** и **netsum**, инициализация сети осуществляется функцией **initnw**.

Аргументы функции:

PR — $R \times 2$ матрица минимальных и максимальных значений R входных элементов,

Si — размер i -го скрытого слоя, для NI слоев,

TFi — функция активации нейронов i -го слоя, по умолчанию **'tansig'**.

BTF — функция обучения сети, по умолчанию **'traingd'**.
BLF — функция настройки весов и смещений, по умолчанию **'learnngdm'**.
PF — функция ошибки, по умолчанию **'mse'**.
Пример
 » **P** = [0 1 2 3 4 5 6 7 8 9 10];
 » **T** = [0 1 2 3 4 3 2 1 2 3 4];
 » **net** = newcf([0 10],[5 1], 'tansig' 'purelin'); % Создание новой сети
 » **net.trainParam.epochs** = 50; % Задание количества циклов обучения
 » **net** = train(**net**,**P**,**T**); % Обучение НС
TRAINLM, Epoch 0/50, MSE 7.77493/0,
 Gradient 138.282/1e-010
TRAINLM, Epoch 25/50, MSE 4.01014e-010/0,
 Gradient 0.00028557/1e-010
TRAINLM, Epoch 50/50, MSE 1.13636e-011/0,
 Gradient 1.76513e-006/1e-010
TRAINLM, Maximum epoch reached, performance goal was not met.
 » **Y** = sim(**net**,**P**); % Использование НС
 » **plot**(**P**,**T**,**P**,**Y**, 'okv') % Графическая иллюстрация работы сети

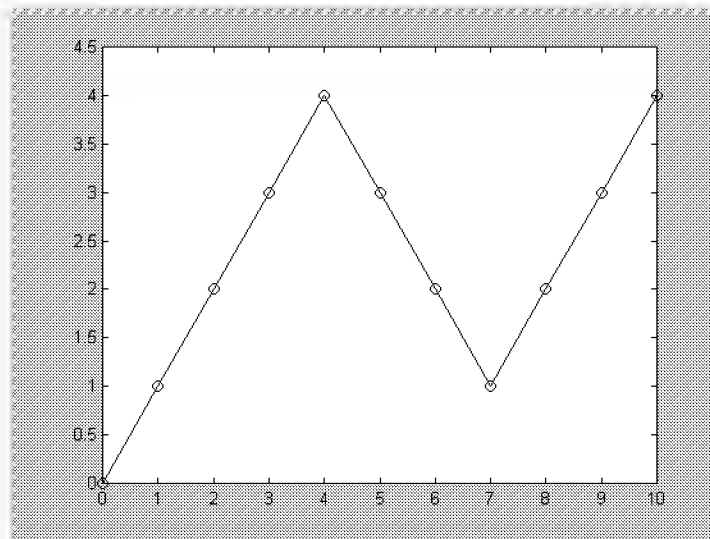


Рис. 4.1. Иллюстрация работы сети

На рис. 4.1 точками отображены элементы обучающей выборки, линией — выход сети.

net = newelm(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) — функция создания сети Элмана. Аргументы — такие же, как и у предыдущей функции.

net = newff(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) — функция создания «классической» многослойной НС с обучением по методу обратного распространения ошибки.

net = newfftd(PR,ID,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) — то же, что и предыдущая функция, но с наличием задержек по входам. Дополнительный аргумент **ID** — вектор входных задержек.

net = newgrnn(P,T,spread) — функция создания обобщенно-регрессионной сети. Аргументы:

P — $R \times Q$ матрица Q входных векторов,

T — $S \times Q$ матрица Q целевых векторов,

spread — отклонение (по умолчанию 1.0).

net = newhop(T) — функция создания сети Хопфилда. Использует только один аргумент.

T — $R \times Q$ матрица Q целевых векторов (значения элементов должны быть +1 или -1).

net = newlin(PR,S,ID,LR) — функция создания слоя линейных нейронов. Аргументы:

PR — $R \times 2$ матрица минимальных и максимальных значений для R входных элементов,

S — число элементов в выходном векторе,

ID — вектор входной задержки (по умолчанию [0]),

LR — коэффициент обучения (по умолчанию 0.01).

Возвращается новый линейный слой.

При записи в форме **net = newlin(PR,S,0,P)** используется аргумент

P — матрица входных векторов,

возвращается линейный слой с максимально возможным коэффициентом обучения при заданной **P**.

net = newlind(P,T) — функция проектирования нового линейного слоя. Данная функция по матрицам входных и выходных векторов методом наименьших квадратов определяет веса и смещения линейной НС.

net = newlvq(PR,S1,PC,LR,LF) — функция создания сети встречного распространения.

Аргументы:

PR — $R \times 2$ матрица минимальных и максимальных значений R входных элементов,

S1 — число скрытых нейронов,

PC — S2 элементов вектора, задающих доли принадлежности к различным классам,
LR — коэффициент обучения, по умолчанию 0.01,
LF — функция обучения, по умолчанию **'learnlv2'**.
net = newp(PR,S,TF,LF) — функция создания персептрона.
 Аргументы:
PR — $R \times 2$ матрица минимальных и максимальных значений R входных элементов,
S — число нейронов,
TF — функция активации, по умолчанию **'hardlim'**,
LF — функция обучения, по умолчанию **'learnp'**.
net = newpnn(P,T,spread) — функция создания вероятностной НС. Аргументы — как у функции **newgrnn**.
net = newrb(P,T,goal,spread) — функция создания сети с радиальными базисными элементами. Аргументы **P**, **T**, **spread** такие же, как у функции **newgrnn**; аргумент **goal** — заданная среднеквадратичная ошибка.
net = newrbe(P,T,spread) — функция создания сети с радиальными базисными элементами с нулевой ошибкой на обучающей выборке.
net = newsom(PR,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR,TND) — функция создания самообучающейся карты с аргументами:
PR — $R \times 2$ матрица минимальных и максимальных значений R входных элементов,
I — размеры i -го слоя, по умолчанию [5 8],
TFCN — топологическая функция, по умолчанию **'hextop'**,
DFCN — функция расстояния, по умолчанию **'linkdist'**,
OLR — коэффициент обучения фазы упорядочивания, по умолчанию 0.9,
OSTEPS — число шагов фазы упорядочивания, по умолчанию 1000,
TLR — коэффициент обучения фазы настройки, по умолчанию 0.02,
TND — расстояние для фазы настройки, по умолчанию 1.

4.2.7. Функции преобразования входов сети. Функции данной группы преобразуют значения входов с использованием операций умножения или суммирования.

netprod(Z1,Z2,...) — возвращает матрицу, элементы которой определяются как произведения элементов входных векторов и смещений. Аргументы **Z1,Z2,...** — матрицы, чьи столбцы ассоциированы с входами или смещениями.

П р и м е р ы

```
» z1 = [1 2 4; 3 4 1];
» z2 = [-1 2 2; -5 -6 1];
» n = netprod(z1,z2)
n =
    -1     4     8
   -15    -24     1
» b = [0; -1];
» n = netprod(z1,z2,concur(b,3)) % Функция concur(b,3) создает 3 ко-
пи вектора смещения
n =
     0     0     0
    15    24    -1
```

netsum(Z1,Z2,...) — то же, что в предыдущем случае, но вместо умножения используется суммирование.

dnetprod(Z,N) возвращает матрицу значений первой производной входов, преобразованных функцией $N = \text{netprod}(Z1,Z2,...)$.

П р и м е р

```
» Z1 = [0; 1; -1];
» Z2 = [1; 0.5; 1.2];
» N = netprod(Z1,Z2)
N =
     0
    0.5000
   -1.2000
» dN_dZ2 = dnetprod(Z2,N)
dN_dZ2 =
     0
     1
    -1
```

dnetsum(Z,N) то же, что и в предыдущем случае, но по отношению к функции **netsum(Z1,Z2,...)**.

4.2.8. Функции весов и расстояний

boxdist(pos) функция определения box-расстояния между нейронами в слое. Имеет один аргумент **pos**-матрицу размера $N \times S$, элементы которой определяют координаты нейронов, возвращает матрицу размера $S \times S$ расстояний. Расстояния (элементы возвращаемой матрицы) вычисляются по выражению:

$D_{ij} = \max(\text{abs}(P_i - P_j))$, где P_i и P_j — векторы, содержащие координаты нейронов i и j .

Пример

» `pos = rand(3,4)` % Случайное размещение 4-х нейронов в 3-мерном пространстве (генерация случайной матрицы 3×4)

```
pos =
    0.8600    0.4966    0.6449    0.3420
    0.8537    0.8998    0.8180    0.2897
    0.5936    0.8216    0.6602    0.3412
```

» `d = boxdist(pos)`

```
d =
     0    0.3635    0.2151    0.5639
    0.3635     0    0.1614    0.6100
    0.2151    0.1614     0    0.5282
    0.5639    0.6100    0.5282     0
```

dist — функция вычисления евклидова расстояния.

Запись:

Z = dist(W,P) — возвращает матрицу, элементы которой являются евклидовыми расстояниями между строками (векторами) матрицы **W** и столбцами матрицы **P** (матрицы должны иметь соответствующие размеры).

D = dist(pos) — в такой форме функция аналогична функции **boxdist(pos)** за тем исключением, что возвращается матрица евклидовых расстояний.

negdist(W,P) — функция идентична предыдущей, но элементы возвращаемой матрицы являются евклидовыми расстояниями, взятыми со знаком минус.

mandist(W,P) — функция аналогична предыдущей, но элементы возвращаемой матрицы являются расстояниями по Манхэттену, которое для векторов **x** и **y** определяется соотношением: $D = \text{sum}(\text{abs}(\mathbf{x} - \mathbf{y}))$.

linkdist(pos) — функция определения линейного расстояния между нейронами в слое. Аналогична функции **boxdist**, отличаясь от последней алгоритмом определения расстояния:

$D_{ij} = 0$, если $i = j$;

$D_{ij} = 1$, если евклидово расстояние между P_i и P_j меньше или равно 1;

$D_{ij} = 2$, если существует k такое, что $D_{ik} = D_{kj} = 1$;

$D_{ij} = 3$, если существуют k_1 и k_2 такие, что $D_{ik_1} = D_{k_1k_2} = D_{k_2j} = 1$;

$D_{ij} = N$, если существуют $k_1, k_2, \dots, k_N$ такие, что $D_{ik_1} = D_{k_1k_2} = \dots = D_{k_Nj} = 1$;

$D_{ij} = S$, если не выполнено ни одно из предыдущих условий.

dotprod(W,P) — функция придания входам **P** некоторых весов **W**. Возвращает матрицу $\mathbf{Z} = \mathbf{W} * \mathbf{P}$.

normprod(W,P) — функция аналогична предыдущий, но каждый элемент возвращаемой матрицы дополнительно делится на сумму элементов соответствующего столбца-сомножителя матрицы **P**.

4.2.9. Функции размещения нейронов (топологические функции). Функции данной группы используются при создании самоорганизующихся карт.

gridtop(dim1,dim2,...,dimN) — функция размещения **N** слоев нейронов в узлах регулярной прямоугольной **N**-мерной решетки. **dim1,dim2,...,dimN** — число нейронов в слоях. Возвращает матрицу, содержащую **N** строк и **(dim1×dim2×...×dimN)** столбцов с координатами нейронов.

Пример

» pos = gridtop(2,3)

pos =

0	1	0	1	0	1
0	0	1	1	2	2

hextop(dim1,dim2,...,dimN) — функция аналогична предыдущей, но размещение нейронов производится в узлах гексагональной (шестиугольной) решетки.

Пример

» pos = hextop(8,5); plotsom(pos)

(см. рис. 4.2).

randtop(dim1,dim2,...,dimN) — аналогична функции **gridtop(dim1,dim2,...,dimN)**, но координаты нейронов выбираются случайным образом.

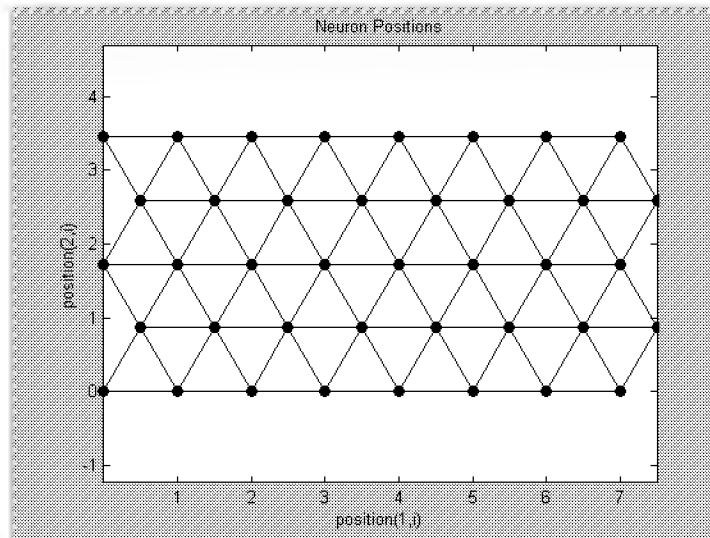
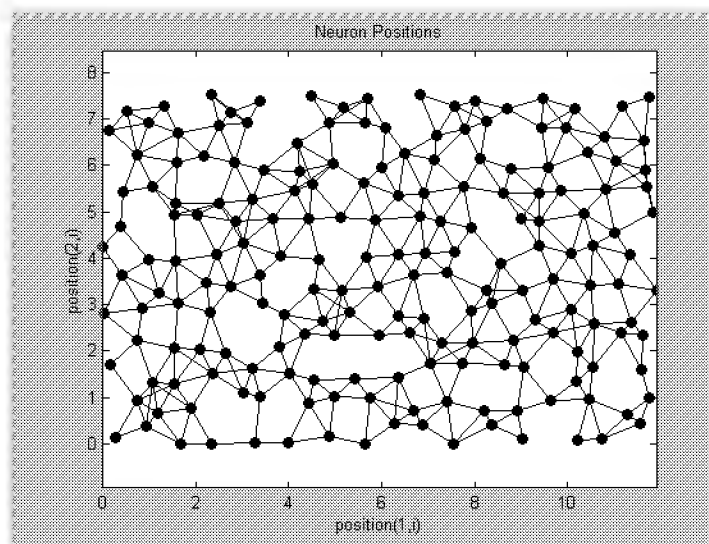
Пример

pos = randtop(16,12); plotsom(pos)

(см. рис. 4.3).

4.2.10. Функции использования нейронных сетей

[Y,Pf,Af] = sim(net,P,Pi,Ai) — функция, моделирующая работу нейронной сети. Аргументы: **net** — имя сети, **P** — ее входы, **Pi** — массив начальных условий входных задержек (по умолчанию они нулевые), **Ai** — массив начальных условий задержек слоя нейронов (по умолчанию они нулевые). Функция возвращает значения выходов **Y** и массивы конечных условий задержек.

Рис. 4.2. Результат выполнения команды `hextop(8,5)`Рис. 4.3. Результат выполнения команды `randtop(16,12)`

Аргументы **Pi, Ai, Pf, Af** используются только в случаях, когда сеть имеет задержки по входам или по слоям нейронов.

Структура данных аргументов:

P — массив размера $N_i \times TS$, каждый элемент которого $P\{i, ts\}$ является матрицей размера $R_i \times Q$.

Pi — массив размера $N_i \times ID$, каждый элемент которого $P_i\{i, k\}$ (i -й вход в момент $ts = k - ID$) является матрицей размера $R_i \times Q$ (по умолчанию — ноль).

Ai — массив размера $N_l \times LD$, каждый элемент которого $A_i\{i, k\}$ (выход i -го слоя в момент $ts = k - LD$) является матрицей размера $S_i \times Q$ (по умолчанию — ноль).

Y — массив размера $N_O \times TS$, каждый элемент которого $Y\{i, ts\}$ является матрицей размера $U_i \times Q$.

Pf — массив размера $N_i \times ID$, каждый элемент которого $P_f\{i, k\}$ (i -й вход в момент $ts = TS + k - ID$) является матрицей размера $R_i \times Q$.

Af — массив размера $N_l \times LD$, каждый элемент которого $A_f\{i, k\}$ (выход i -го слоя в момент $ts = TS + k - LD$) является матрицей размера $S_i \times Q$, при этом

Ni = net.numInputs — количество входов сети,

Nl = net.numLayers — количество ее слоев,

No = net.numOutputs — количество выходов сети,

ID = net.numInputDelays — входные задержки,

LD = net.numLayerDelays — задержки слоя,

TS = Number of time steps — число временных интервалов,

Q = Batch size — размер набора подаваемых векторов,

Ri = net.inputs{ i }.size — размер i -го вектора входа,

Si = net.layers{ i }.size — размер i -го слоя,

Ui = net.outputs{ i }.size — размер i -го вектора выхода.

net = init(net) — функция инициализирует нейронную сеть с именем **net**, устанавливая веса и смещения сети в соответствии с установками **net.initFcn** и **net.initParam**.

[net, Y, E, Pf, Af] = adapt(net, P, T, Pi, Ai) — функция адаптации НС. Выполняет адаптацию сети в соответствии с установками **net.adaptFcn** и **net.adaptParam**. Здесь **E** — ошибки сети, **T** — целевые значения выходов (по умолчанию — ноль); остальные аргументы — как у команды **sim**.

[net, tr] = train(net, P, T, Pi, Ai) — функция осуществляет обучение НС в соответствии с установками **net.trainFcn** и **net.trainParam**. Здесь **tr** — информация о выполнении процесса обучения (количество циклов и соответствующая ошибка обучения).

disp(net) функция возвращает развернутую информацию о структуре и свойствах НС.

П р и м е р

```
» net = newp([-1 1; 0 2],3); % Создание НС типа персептрон
```

```
» disp(net)
```

Neural Network object:

architecture:

numInputs: 1

numLayers: 1

biasConnect: [1]

inputConnect: [1]

layerConnect: [0]

outputConnect: [1]

targetConnect: [1]

numOutputs: 1 (read-only)

numTargets: 1 (read-only)

numInputDelays: 0 (read-only)

numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1×1 cell} of inputs

layers: {1×1 cell} of layers

outputs: {1×1 cell} containing 1 output

targets: {1×1 cell} containing 1 target

biases: {1×1 cell} containing 1 bias

inputWeights: {1×1 cell} containing 1 input weight

layerWeights: {1×1 cell} containing no layer weights

functions:

adaptFcn: 'adaptwb'

initFcn: 'initlay'

performFcn: 'mae'

trainFcn: 'trainwb'

parameters:

adaptParam: .passes

initParam: (none)

performParam: (none)

trainParam: .epochs, .goal, .max_fail, .show,
.time

weight and bias values:

IW: {1×1 cell} containing 1 input weight matrix

LW: {1×1 cell} containing no layer weight matrices

b: {1×1 cell} containing 1 bias vector

other:

userdata: (user stuff)

display(net) — то же, что предыдущая команда, но дополнительно возвращает имя нейронной сети.

4.2.11. Графические функции

hintonw(W,maxw,minw) — функция возвращает так называемый хинтоновский график матрицы весов, при котором каждый весовой коэффициент отображается квадратом с площадью, пропорциональной величине данного коэффициента. Знак отображается цветом квадрата.

Аргументы:

W — матрица весов, **maxw** и **minw** — минимальное и максимальные значения ее коэффициентов (могут не задаваться).

Пример

» **W = randi(2,3)**

W =

-0.8842 0.6263 -0.7222

-0.2943 -0.9803 -0.5945

» **hintonw(W)**

(см. рис. 4.4).

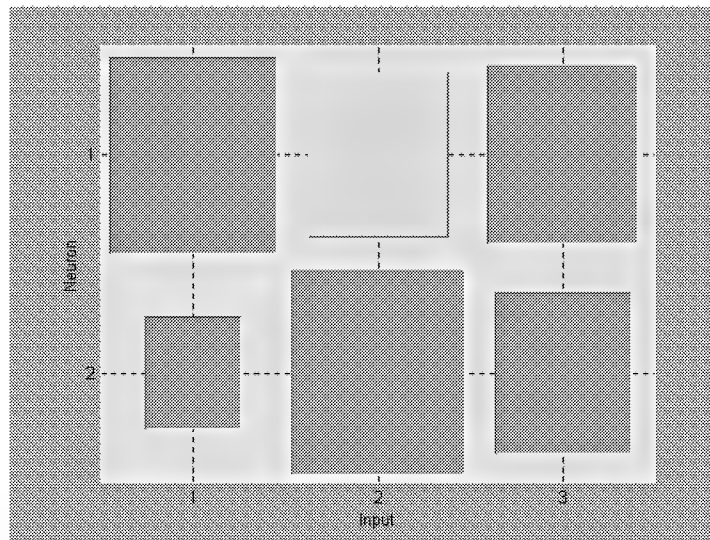


Рис. 4.4. Иллюстрация к выполнению функции **hintonw**

hintonwbm(W,b,maxw,minw) — то же, что и предыдущая функция, но на графике отображаются не только веса, но и смещения.

plotbr(tr,name,epoch) — функция возвращает графики изменения критерия качества НС в процессе обучения при использовании Байесовского метода (см. выше).

Аргументы:

tr — запись процесса обучения, **name** — имя НС, **epoch** — количество циклов обучения (по умолчанию — длина записи обучения).

Пример

```
» p = [-1:.05:1];
» t = sin(2*pi*p)+0.1*randn(size(p));
» net = newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
% Создание новой сети
» [net,tr] = train(net,p,t); % Обучение сети
TRAINBR, Epoch 0/100, SSE 228.933/0, SSW 21461.7,
Grad 2.33e+002/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 25/100, SSE 0.235423/0, SSW 211.044,
Grad 9.43e-002/1.00e-010, #Par 1.35e+001/61
TRAINBR, Epoch 50/100, SSE 0.240881/0, SSW 121.647,
Grad 1.87e-001/1.00e-010, #Par 1.23e+001/61
TRAINBR, Epoch 75/100, SSE 0.239867/0, SSW 116.884,
Grad 1.62e-002/1.00e-010, #Par 1.22e+001/61
TRAINBR, Epoch 100/100, SSE 0.239762/0, SSW 116.871,
Grad 9.60e-003/1.00e-010, #Par 1.22e+001/61
TRAINBR, Maximum epoch reached.
» plotbr(tr)
(см. рис. 4.5).
```

plotep(w,b,e) — функция отображает позиции весов и смещений на поверхности ошибки НС.

Аргументы:

w, **b**, **e** — соответственно, матрицы весов, смещений и ошибок. Возвращается вектор, используемый для продолжения графика, созданного функцией **plotes** (см. ниже).

plotes(wv,bv,e,v) — функция возвращает график поверхности ошибки однослойного нейрона.

Аргументы:

wv, **bv** — соответственно, наборы значений веса и смещения нейрона, **e** — матрица значений ошибки, **v** — опция вида изображения (по умолчанию, $[-37.5, 30]$). Использование функции иллюстрирует рис. 4.8 (см. ниже).

plotpc(W,b) — функция возвращает график линии решения для персептрона.

Аргументы:

W — матрица весов, **b** — вектор смещений. Используется совместно с функцией **plotpv** (см. ниже).

plotperf(tr,goal,name,epoch) — возвращает график изменения критерия качества НС в процессе обучения. Аргументы: **tr** — за-

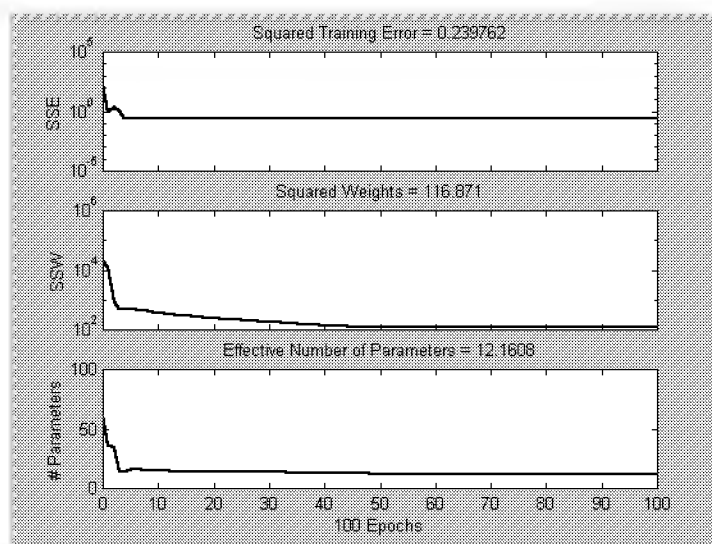


Рис. 4.5. Результат выполнения функции **plotbr(tr)**

пись процесса обучения, **goal** — целевое значение критерия, **name** — имя НС, **epoch** — количество циклов обучения.

plotpv(p,t) — функция возвращает графическое отображение входных **p** и целевых **t** векторов персептрона.

Пример

» **p** = [0 0 1 1; 0 1 0 1];

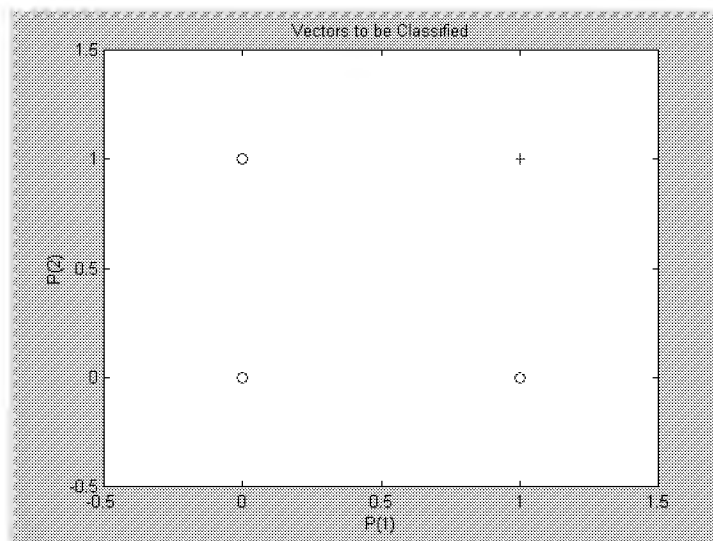
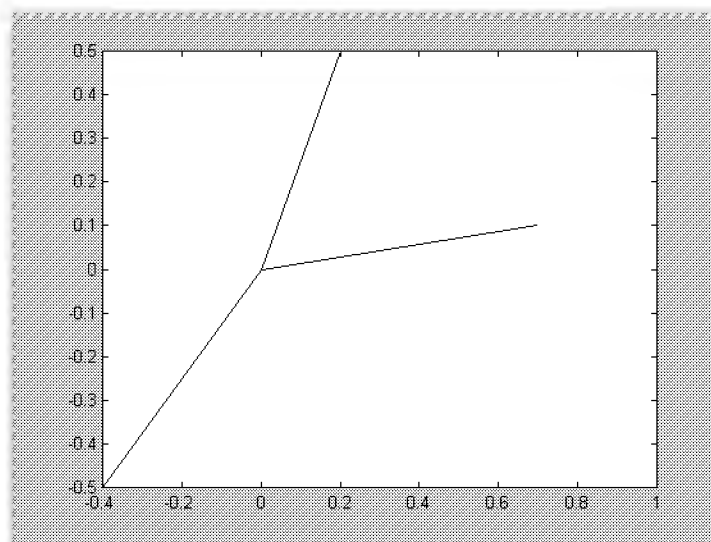
» **t** = [0 0 0 1];

» **plotpv(p,t)**

(см. рис. 4.6).

plotsom(pos) — функция возвращает графическое представление расположения нейронов в самоорганизующихся картах (см. Функции размещения нейронов, рис. 4.2, 4.3).

plotv(M,t) — функция графического изображения векторов.

Рис. 4.6. Результат использования функции `plotpv(p,t)`Рис. 4.7. Результат использования функции `plotv(M,t)`

Аргументы:

M — матрица с двумя строками, столбцы которой ассоциированы с векторами, **t** — опция, задающая тип линии.

Пример

```
» plotv([-4 0.7 .2; -0.5 .1 0.5],'-')
```

(см. рис. 4.7).

plotvec(M,C,m) — функция графического изображения векторов разными цветами.

Аргументы:

M — матрица с двумя строками, столбцы которой ассоциированы с векторами, **C** — строка задания цветов, **m** — тип точек, указывающих концы векторов (по умолчанию +).

4.2.12. Прочие функции

errsurf(p,t,wv,bv,f) — функция, возвращающая матрицу значений поверхности ошибок нейрона с одним входом и одним выходом в зависимости от значений веса и смещения. Аргументы:

p — вектор значений входа,.

t — вектор значений выхода,.

wv — набор значений веса нейрона,.

bv — набор значений смещения.

f — название реализуемой функции активации (строка).

Размер возвращаемой матрицы = (количество значений **bv**) × (количество значений **wv**).

Пример

```
» p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
```

```
» t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
```

```
» wv = -1:1:1; bv = -2.5:2.5:2.5;
```

```
» es = errsurf(p,t,wv,bv,'logsig');
```

```
» plotes(wv,bv,es,[60 30])
```

(см. рис. 4.8)

maxlinlr(P) — возвращает максимальную величину коэффициента обучения для линейного слоя нейронов. Здесь **P** — матрица входов.

При записи в форме **maxlinlr(P,'bias')** функция возвращает максимальную величину коэффициента обучения для линейного слоя нейронов со смещением.

gensim(net,st) — функция генерирует нейросетевой блок Simulink (см. рис. 4.9) для последующего моделирования НС средствами этого пакета.

Пример

```
» net = newff([0 1],[5 1]); % Создание новой НС  
» gensim(net)
```

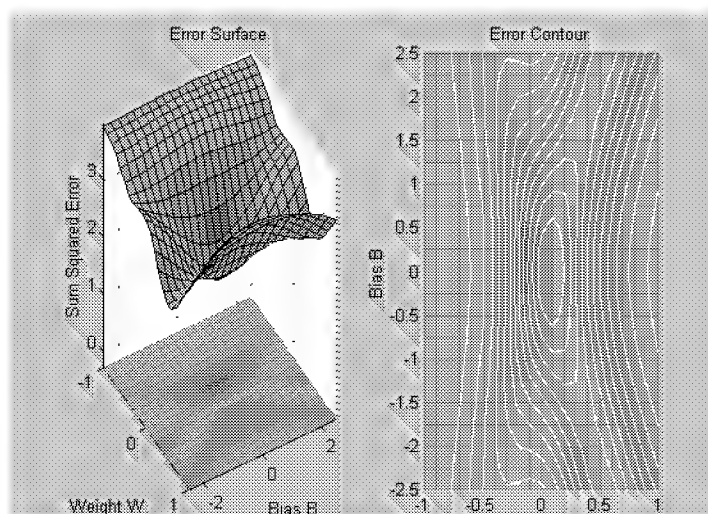


Рис. 4.8. Иллюстрация к примеру выполнения функции **errsurf**

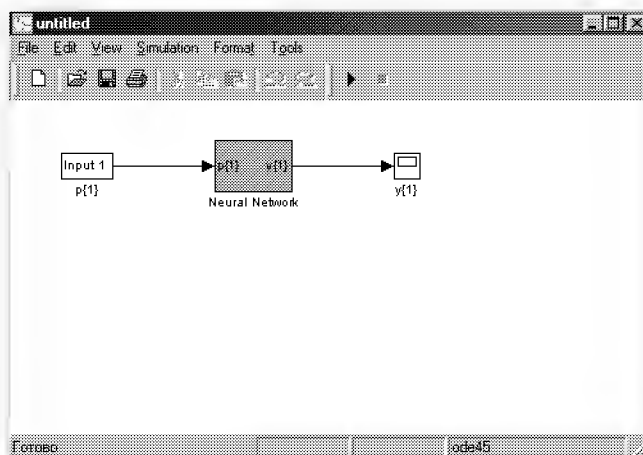


Рис. 4.9. Результат выполнения функции **gensim**

initlay(net) — функция инициализации слоев нейронной сети. В качестве аргумента использует имя (идентификатор) **net** НС. Возвращает нейронную сеть, слои нейронов в которой инициализированы в соответствии с функцией **net.layers{ i }.initFcn**. В форме **initlay(code)**, где строковая переменная **code** может принимать значения **'pnames'** или **'pdefaults'** функция возвращает информацию о именах или о значениях по умолчанию параметров инициализации.

initnw(net,i) — функция инициализации слоя i . Возвращает НС, веса и смещения в i -м слое которой обновлены в соответствии с алгоритмом инициализации Nguyen-Widrow (так, что зоны «влияния» каждого нейрона в слое распределены равномерно).

initwb(net,i) — почти то же, что в предыдущем случае, но веса и смещения i -го слоя инициализируются в соответствии с их собственными функциями инициализации.

ddotprod — функция определения производной от результата **Z** умножения матрицы весов **W** на матрицу входов **P**.

Запись:

```
dZ_dP = ddotprod('p',W,P,Z)
dZ_dW = ddotprod('w',W,P,Z)
```

Примеры

```
» W = [0 -1 0.2; -1.1 1 0];
» P = [0.1; 0.6; -0.2];
» Z = dotprod(W,P) % Вычисление произведения Z=W*P
Z =
    -0.6400
     0.4900
» dZ_dP = ddotprod('p',W,P,Z)
dZ_dP =
     0    -1.0000    0.2000
    -1.1000    1.0000     0
» dZ_dW = ddotprod('w',W,P,Z)
dZ_dW =
     0.1000
     0.6000
    -0.2000
```

4.3. Примеры создания и использования нейронных сетей

4.3.1. Нейронные сети для аппроксимации функций. Создадим обобщенно-регрессионную НС с именем **a** для аппроксимации функции вида

$y = x^2$ на отрезке $[-1, 1]$, используя следующие экспериментальные данные:

```
x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1],
y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1].
```

Процедура создания и использования данной НС описывается следующим образом:

```
» P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1]; % Задание входных значений
» T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1]; % Задание
выходных значений
» a = newgrnn(P,T,0.01); % Создание НС с отклонением 0.01
» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8]) % Опрос НС
Y =
    0.8200    0.6400    0.0400    0.0900    0.8100
```

Как видно, точность аппроксимации в данном случае получилась не очень высокой.

Можно попытаться улучшить качество аппроксимации за счет подбора величины отклонения, но в условиях примера приемлемый результат легко достигается путем применения сети с радиальными базисными элементами:

```
» a = newrbf(P,T);
» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8]) % Опрос НС
Y =
    0.8100    0.4900    0.0900    0.1600    0.6400
```

Созданную сеть можно сохранить для последующего использования набором в командной строке **save('a')**; при этом будет создан файл **a.mat**, т.е. файл с именем НС и расширением **mat**. В последующих сеансах работы сохраненную сеть можно загрузить, используя функцию **load('a')**. Естественно, допустимы все другие формы записи операторов **save** и **load**.

Рассмотрим теперь аналогичную задачу, но с использованием линейной НС.

Пусть экспериментальная информация задана значениями:

```
x = [+1.0 +1.5 +3.0 -1.2],
y = [+0.5 +1.1 +3.0 -1.0].
```

Процесс создания, обучения и использования линейной НС с именем **b** иллюстрируется приведенными функциями и рис. 4.10.

```
» P = [+1.0 +1.5 +3.0 -1.2];
» T = [+0.5 +1.1 +3.0 -1.0];
» maxlr = maxlinlr(P,'bias'); % Определение величины
коэффициента обучения
» b = newlin([-2 2],1,[0],maxlr); % Создание линейной НС с
именем b
» b.trainParam.epochs = 15; % Задание количества циклов
обучения
» b = train(b,P,T); % Обучение НС
TRAINWB, Epoch 0/15, MSE 2.865/0.
TRAINWB, Epoch 15/15, MSE 0.0730734/0.
```

TRAINWB, Maximum epoch reached.

» p = -1.2;

» y = sim(b,p) % Опрос сети

y =

-1.1803

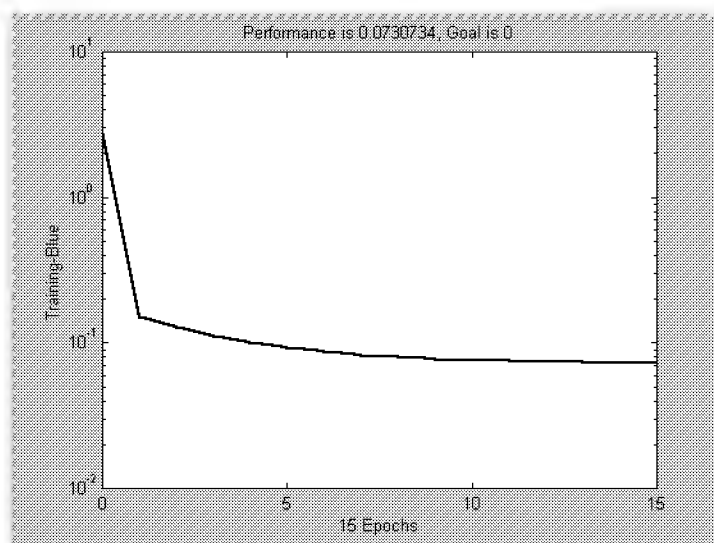


Рис. 4.10. Изменение ошибки сети в процессе ее обучения

4.3.2. Прогнозирование значений процесса. Рассмотрим теперь такой пример. Предположим, что имеется сигнал (функция времени), описываемый соотношением $x(t) = \sin(4\pi t)$, который подвергается дискретизации с интервалом 0.025 с.

Построим линейную нейронную сеть, позволяющую прогнозировать будущее значение подобного сигнала по 5 предыдущим. Решение данной задачи иллюстрируется ниже.

» t = 0:0.025:5; % Задание диапазона времени от 0 до 5 секунд

» x = sin(t*4*pi); % Предсказываемый сигнал

» Q = length(x);

» % Создание входных векторов

» P = zeros(5,Q); % Создание нулевой матрицы P

» P(1,2:Q) = x(1,1:(Q-1));

» P(2,3:Q) = x(1,1:(Q-2));

» P(3,4:Q) = x(1,1:(Q-3));

» P(4,5:Q) = x(1,1:(Q-4));

» P(5,6:Q) = x(1,1:(Q-5));

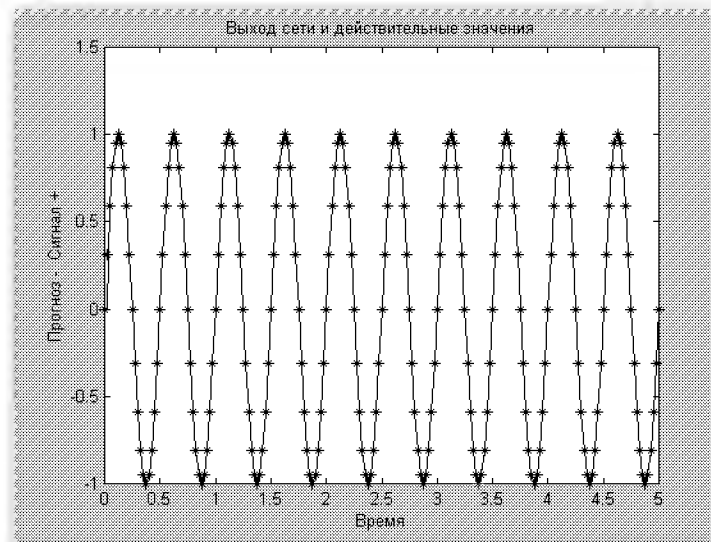


Рис. 4.11. Исходный сигнал и прогноз

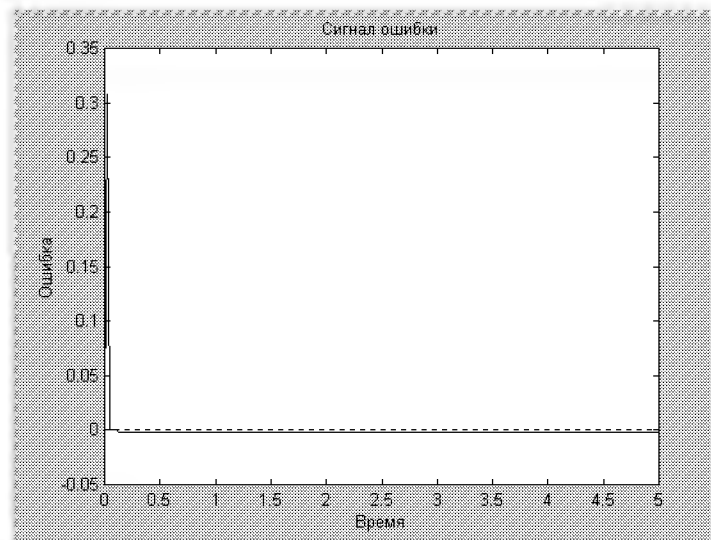


Рис. 4.12. Ошибка прогноза

```

» s = newlind(P,x); % Создание новой НС с именем s
» y = sim(s,P); % Расчет прогнозируемых значений
» % Создание графиков исходного сигнала и прогноза
» plot(t,y,t,x,'*');
» xlabel('Время');
» ylabel('Прогноз - Сигнал +');
» title('Выход сети и действительные значения');
» % Расчет и создание графика ошибки прогноза
» e = x-y;
» plot(t,e)
» hold on
» plot([min(t) max(t)],[0 0], 'r')
» hold off
» xlabel('Время');
» ylabel('Ошибка');
» title('Сигнал ошибки');

```

В данном случае сеть создавалась с помощью функции `newlind`, при которой не требуется дополнительного обучения. Судя по графикам результатов, приведенных на рис. 4.11 и 4.12, точность прогноза с использованием линейной НС можно считать хорошей.

4.3.3. Использование слоя Кохонена. Рассмотрим задачу автоматического выявления (в режиме обучения без учителя) центров кластеров входов для двумерного случая с использованием слоя Кохонена (слоя «состязывающихся» нейронов). Решение данной задачи приведено ниже.

```

» X = [0 1; 0 1]; % Задание диапазонов возможного положения центров
кластеров
» % Задание параметров для моделирования исходных данных,
» % принадлежащих 8 классам (кластерам)
» clusters = 8;
» points = 10;
» std_dev = 0.05;
» P = nngenc(X,clusters,points,std_dev); % Моделирование
входных данных
» h = newc([0 1;0 1],8,.1); % Создание слоя Кохонена
» h.trainParam.epochs = 500; % Задание количества циклов
обучения
» h = init(h); % Инициализация сети
» h = train(h,P); % Обучение сети
» w = h.IW{1};
» % Вывод графика исходных данных и выявленных центров кла-
стеров
» plot(P(1,:),P(2:,:),'+r');
» hold on; plot(w(:,1),w(:,2),'ob');
» xlabel('p(1)');

```

```

» ylabel('p(2)');
» p = [0; 0.2]; % Задание нового входного вектора
» y = sim(h,p) % Опрос сети
» y =
    (3,1)    1

```

Работу обученной сети иллюстрирует рис. 4.13 и результат ее опроса (который выдается в форме разреженной матрицы). В усло-

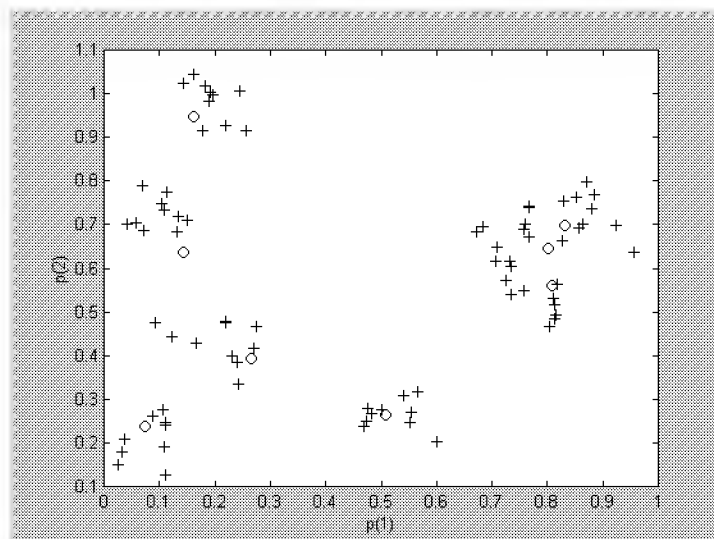


Рис. 4.13. Исходные данные и выявленные центры кластеров

виях примера предъявленный вектор отнесен к третьему классу (кластеру).

4.3.4. Сеть Хопфилда с двумя нейронами. Рассмотрим сеть Хопфилда, имеющую два нейрона и два устойчивых состояния, отображаемых векторами $[1 \ -1]$ и $[-1 \ 1]$. Представим эти векторы с помощью рис. 4.14, выводимого программой

```

» T = [+1 -1; -1 +1];
» plot(T(1,:),T(2,:), 'r*')
» axis([-1.1 1.1 -1.1 1.1]);
» title('Пространство векторов НС Хопфилда');
» xlabel('a(1)');
» ylabel('a(2)');

```

Создадим сеть Хопфилда (с именем Н) и проверим ее работу, подав на вход векторы, соответствующие устойчивым точкам. Если

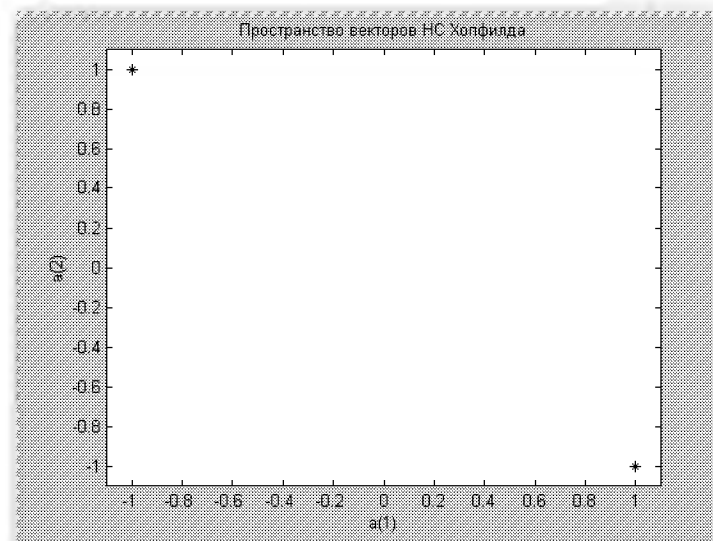


Рис. 4.14. Устойчивые точки сети Хопфилда

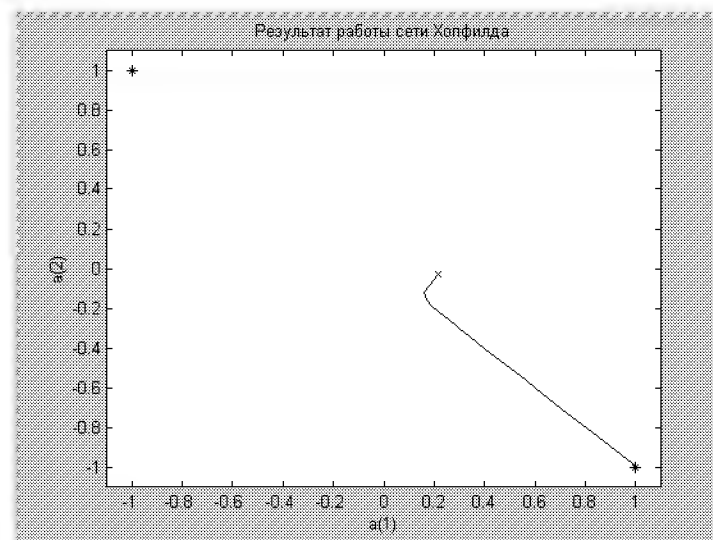


Рис. 4.15. Результат работы сети Хопфилда

сеть работает правильно, она должна выработать эти же векторы без каких-либо изменений.

```
» H=newhop(T); % Создание НС Хопфилда
» [Y,Pf,Af]=sim(H,2,[],T);Y % Опрос сети Хопфилда
Y =
    1    -1
   -1     1
```

Как видно из результата опроса, сеть работает правильно. Подадим теперь на ее вход произвольный вектор.

```
» a={rands(2,1)}; % Задание случайного вектора
» [y,Pf,Af]=sim(H,1 50,{ },a);
» plot(T(1,:),T(2,:),'r*')
» axis([-1.1 1.1 -1.1 1.1]);
» record=[cell2mat(a) cell2mat(y)];
» start=cell2mat(a);
» hold on;
» plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
» xlabel('a(1)'); ylabel('a(2)');
» title('Результат работы сети Хопфилда');
```

Результат иллюстрируется рис. 4.15.

4.3.5. Классификация с помощью персептрона. Следующий пример иллюстрирует решение задачи классификации с помощью персептрона. Исходные входные векторы (с указанием их принадлежности к одному из двух классов) и результат настройки персептрона (с именем **My_net**) представлены на рис. 4.16.

```
» % Задание входных векторов с указанием их принадлежности
» % одному из двух классов
» P=[-0.5 -0.5 +0.3 -0.1;-0.5 +0.5 -0.5 +1.0];
» T=[1 1 0 0];
» plotpv(P,T); % Графическое представление исходных
векторов
» % Создание персептрона с указанием границ изменений
входов и 1 нейроном
» My_net=newp([-1 1;-1 1],1);
» E=1;
» My_net=init(My_net); % Инициализация персептрона
» % Организация цикла адаптивной настройки персептрона
» % с выводом графика разделяющей линии
» while (sse(E))
    [My_net,Y,E]=adapt(My_net,P,T);
```

```
linehandle = plotpc(My_net.IW{1},My_net.b{1});
drawnow;
end;
```

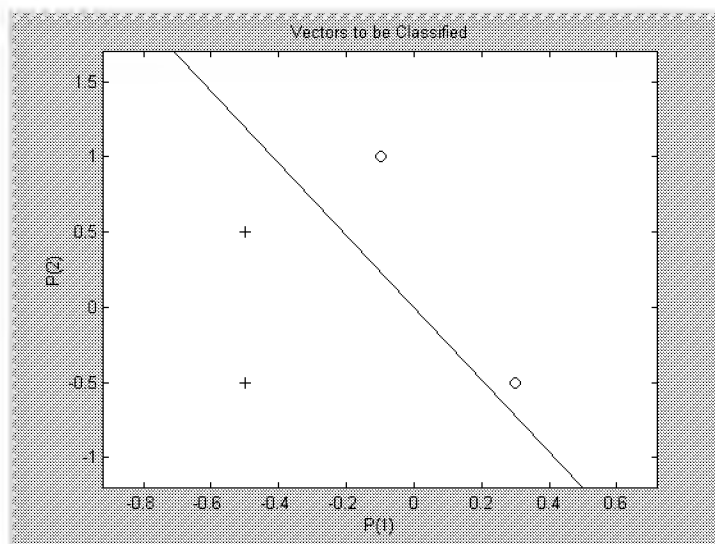


Рис. 4.16. Исходные входные векторы и разделяющая линия

4.3.6. Адаптивный линейный прогноз. В предыдущем примере настройка НС производилась адаптивно. Отличие такой настройки от выполняемой, например, с помощью метода обратного распространения ошибки, заключается в том, что векторы обучающей выборки поступают на вход сети не все «одновременно», а последовательно, по одному, при этом после предъявления очередного вектора производится корректировка весов и смещений и может быть произведен опрос сети, затем все повторяется. Адаптивная настройка особенно удобна при работе НС в «реальном» масштабе времени.

Рассмотрим пример задачи с прогнозированием значений сигнала (по 5 предыдущим значениям) с использованием указанной настройки.

Предположим, что исходный сигнал определен на интервале времени от 0 до 6 с, при этом при $0 \leq t < 4$ с он описывается соотношением $x(t) = \sin(4\pi t)$, а при $4 \leq t \leq 6$ с — соотношением $x(t) = \sin(8\pi t)$. График такого сигнала приведен на рис. 4.17.

```
» time1=0:0.05:4; % от 0 до 4 секунд
» time2=4.05:0.024:6; % от 4 до 6 секунд
```

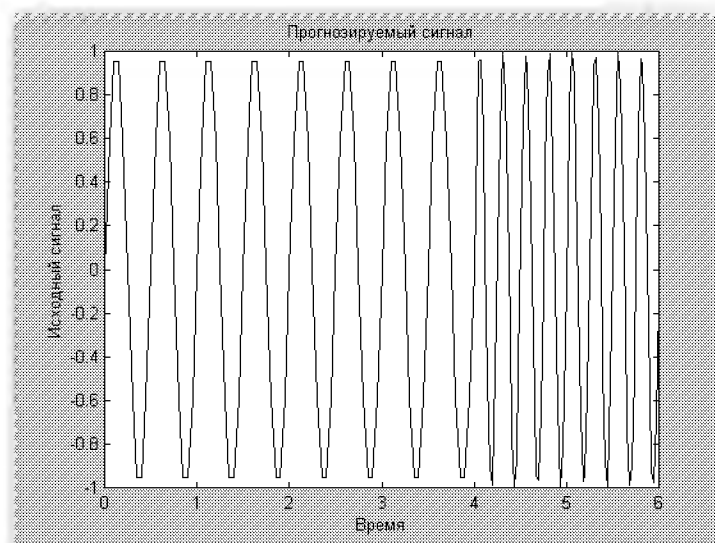


Рис. 4.17. График прогнозируемого сигнала

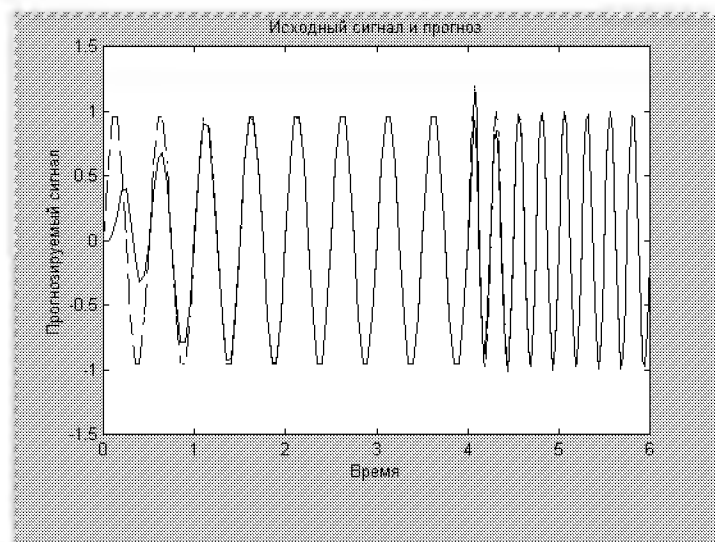


Рис. 4.18. Исходный сигнал и прогноз

```

» time = [time1 time2];
» % T определяет исходный сигнал:
» T = con2seq([sin(time1*4*pi) sin(time2*8*pi)]);
» % График исходного сигнала:
» plot(time,cat(2,T{:}))
» xlabel('Время');
» ylabel('Исходный сигнал');
» title('Прогнозируемый сигнал');
Для прогноза значений сигнала создадим линейную НС.
» % Входной и целевой прогнозируемый сигнал одинаковы:
» P = T;
» % Задание коэффициента обучения
» lr = 0.1;
» % Для прогноза используются 5 предыдущих значений
» delays = [1 2 3 4 5];
» % Создание и настройка линейной НС
» net = newlin(minmax(cat(2,P{:})),1,delays,lr); % Создание НС
» [net,y,e] = adapt(net,P,T); % Адаптивная настройка сети
» % Графики исходного сигнала и прогноза
» plot(time,cat(2,y{:}),time,cat(2,T{:}),'- -')
» xlabel('Время');
» ylabel('Прогнозируемый сигнал');
» title('Исходный сигнал и прогноз');

```

Исходный сигнал и прогноз для этого примера приведены на рис. 4.18. Как видно из рис. 4.18, полученный результат можно считать удовлетворительным.

4.3.7. Использование сети Элмана. Рассмотрим задачу восстановления формы сигнала с использованием рекуррентной сети Элмана. Пусть имеется два синусоидальных сигнала — один с единичной амплитудой, другой — с амплитудой, равной двум:

```

p1 = sin(1:20);
p2 = sin(1:20)*2.

```

Пусть целевым сигналом будет сигнал, составленный из их амплитудных значений

```

t1 = ones(1,20);
t2 = ones(1,20)*2

```

при этом данные амплитуды чередуются, так что входные и целевые значения могут быть представлены в форме

```

p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2].

```

Преобразуем эти значения в последовательности:

```

Pseq = con2seq(p);
Tseq = con2seq(t).

```

После этого можно непосредственно перейти с проектированию НС. В рассматриваемой случае имеем, очевидно, один вход и один выход, т.е. в сети должны присутствовать один входной элемент и один выходной нейрон. Число нейронов в скрытом слое может быть, вообще говоря, любым (оно зависит от сложности задачи); примем, что этот слой содержит 10 нейронов. Дальнейшее решение задачи иллюстрируется ниже, при этом на рис. 4.19 приведен график изменения ошибки сети в процессе ее обучения, а на рис. 4.20 — результаты тестирования сети.

```

» % Задание исходных данных
» p1 = sin(1:20);
» t1 = ones(1,20);
» p2 = sin(1:20)*2;
» t2 = ones(1,20)*2;
» p = [p1 p2 p1 p2];
» t = [t1 t2 t1 t2];
» Pseq = con2seq(p);
» Tseq = con2seq(t);
» % Создание сети Элмана с диапазоном входа [-2, 2], 10
нейронами
» % скрытого слоя, одним выходным нейроном,
» % функцией активации в виде гиперболического тангенса
» % для нейронов скрытого слоя, линейной функцией
активации
» % для выходного нейрона, функцией обучения с адаптацией
» % коэффициента обучения
» net = newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx');
» % Задание параметров обучения:
» net.trainParam.epochs = 500; % Число циклов обучения
» net.trainParam.goal = 0.01; % Целевое значение функции
ошибки
» net.performFcn = 'sse'; % Задание вида функции ошибки
» % Обучение сети. По умолчанию промежуточные результаты обуче-
ния
» % выводятся через 25 циклов
» [net,tr] = train(net,Pseq,Tseq);
TRAINGDX, Epoch 0/500, SSE 443.798/0.01,
Gradient 387.439/1e-006
TRAINGDX, Epoch 25/500, SSE 22.1356/0.01,
Gradient 7.76533/1e-006
TRAINGDX, Epoch 50/500, SSE 20.0141/0.01,
Gradient 2.17503/1e-006

```

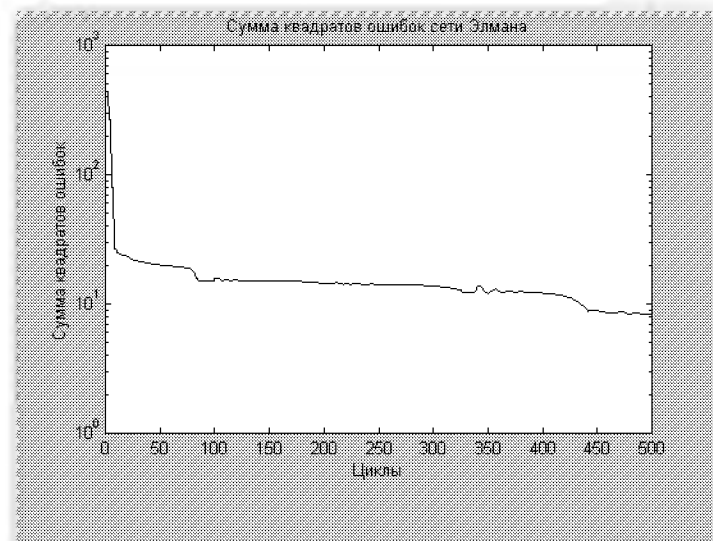


Рис. 4.19. Изменение ошибки сети в процессе обучения

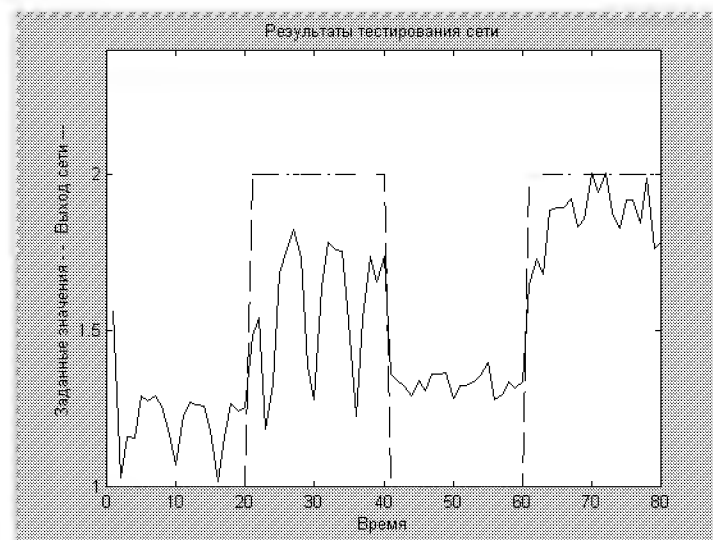


Рис. 4.20. Результаты тестирования сети

```
    TRAINGDX, Epoch 75/500, SSE 18.9825/0.01,  
Gradient 1.26454/1e-006  
    TRAINGDX, Epoch 100/500, SSE 15.8484/0.01,  
Gradient 9.11439/1e-006  
    TRAINGDX, Epoch 125/500, SSE 15.1932/0.01,  
Gradient 2.92454/1e-006  
    TRAINGDX, Epoch 150/500, SSE 15.1337/0.01,  
Gradient 2.72534/1e-006  
    TRAINGDX, Epoch 175/500, SSE 14.9634/0.01,  
Gradient 2.75817/1e-006  
    TRAINGDX, Epoch 200/500, SSE 14.3391/0.01,  
Gradient 3.06463/1e-006  
    TRAINGDX, Epoch 225/500, SSE 14.2136/0.01,  
Gradient 3.24916/1e-006  
    TRAINGDX, Epoch 250/500, SSE 14.1567/0.01,  
Gradient 3.26074/1e-006  
    TRAINGDX, Epoch 275/500, SSE 14.0799/0.01,  
Gradient 3.1542/1e-006  
    TRAINGDX, Epoch 300/500, SSE 13.7704/0.01,  
Gradient 3.27526/1e-006  
    TRAINGDX, Epoch 325/500, SSE 12.6771/0.01,  
Gradient 3.72479/1e-006  
    TRAINGDX, Epoch 350/500, SSE 12.1381/0.01,  
Gradient 5.86736/1e-006  
    TRAINGDX, Epoch 375/500, SSE 12.315/0.01,  
Gradient 3.92575/1e-006  
    TRAINGDX, Epoch 400/500, SSE 12.1414/0.01,  
Gradient 3.89053/1e-006  
    TRAINGDX, Epoch 425/500, SSE 11.1606/0.01,  
Gradient 3.93421/1e-006  
    TRAINGDX, Epoch 450/500, SSE 8.91345/0.01,  
Gradient 5.55002/1e-006  
    TRAINGDX, Epoch 475/500, SSE 8.56053/0.01,  
Gradient 3.87387/1e-006  
    TRAINGDX, Epoch 500/500, SSE 8.34089/0.01,  
Gradient 3.7055/1e-006  
    TRAINGDX, Maximum epoch reached, performance goal was  
not met.  
    » % Построение графика функции ошибки  
    » semilogy(tr.epoch,tr.perf);  
    » title('Сумма квадратов ошибок сети Элмана');  
    » xlabel('Циклы');  
    » ylabel('Сумма квадратов ошибок');  
    » % Тестирование сети
```



```

» a = sim(net,Pseq);
» time = 1:length(p);
» time = 1:length(p);
» plot(time,t,'-',time,cat(2,a{:}))
» title('Результаты тестирования сети');
» xlabel('Время');
» ylabel('Заданные значения -- Выход сети ---')

```

4.3.8. Задача классификации: применение сети встречного распространения. Предположим, поставлена следующая задача классификации: задан набор из 10 векторов, представленных в виде столбцов матрицы

$$P = \begin{bmatrix} -3 & -2 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 3 \\ 0 & 1 & -1 & 2 & 1 & -1 & -2 & 1 & -1 & 0 \end{bmatrix},$$

а также задан вектор-строка, указывающий принадлежность каждого вектора к одному из двух классов:

$$Tc = [1 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 1 \ 1 \ 1].$$

Требуется: построить автоматический классификатор подобных векторов, используя приведенные данные как обучающую выборку.

Решение подобной задачи проведем с применением сети встречного распространения так, как это показано ниже.

```

» P = [-3 -2 -2 0 0 0 0 +2 +2 +3; 0 +1 -1 +2 +1 -1 -2 +1 -1 0];
» C = [1 1 1 2 2 2 2 1 1 1];
» T = ind2vec(C); % Преобразование вектора C в матрицу T с двумя
строками
» % Создание новой сети встречного распространения требует 4-х па-
раметров:
» % 1) матрицы минимальных и максимальных значений
входных элементов,
» % 2) числа скрытых нейронов,
» % 3) вектор с элементами, указывающими долю каждого из классов,
» % 4) величины коэффициента обучения
» net = newlvq(minmax(P),4,[.6 .4],0.1); % Создание сети
» net.trainParam.epochs = 150; % Задание числа циклов
обучения
» net.trainParam.show = Inf; % Запрет на выдачу
промежуточных результатов
» net = train(net,P,T); % Обучение НС
TRAINWB1, Epoch 150/150
TRAINWB1, Maximum epoch reached.
» Y = sim(net,P) % Тестирование сети
Y =
    1    1    1    0    0    0    0    1    1    1
    0    0    0    1    1    1    1    0    0    0

```

Как видно из результатов тестирования, классификация элементов обучающей выборки произведена точно (три первых и три последних вектора отнесены к первому классу, остальные — ко второму).

4.3.9. Создание и использование самоорганизующейся карты.

Как отмечалось, самоорганизующиеся карты можно рассматривать как усовершенствованную модификацию слоя конкурирующих нейронов (слоя Кохонена). От последнего данный вид НС отличается тем, что:

1) нейроны распределяются некоторым пространственным образом (по одному из трех возможных вариантов: в узлах прямоугольной решетки, гексагональной решетки или случайно);

2) на этапе самообучения корректируются веса не только нейрона-«победителя», но и группы нейронов в его некоторой пространственной окрестности.

Назначения самоорганизующихся карт такое же, как и у слоя Кохонена — выявление в режиме самообучения центров кластеров входных векторов.

Создание и использование самоорганизующейся карты рассмотрим на примере кластеризации двумерных векторов (исходные данные приведены на рис. 4.21).

» `P = rand(2,1000);` % Задание случайных двумерных входных векторов

» `plot(P(1,:),P(2,:),'+r')` % Визуальное изображение входных векторов

» Создание НС с $5 \times 6 = 30$ нейронами; все установки — по умолчанию

» `net = newsom([0 1; 0 1],[5 6]);`

» `net.trainParam.epochs = 1000;` % Задание числа циклов настройки

» `net.trainParam.show = 200;` % Задание периодичности вывода информации

» `net = train(net,P);` % Настройка сети

TRAINWB1, Epoch 0/1000

TRAINWB1, Epoch 200/1000

TRAINWB1, Epoch 400/1000

TRAINWB1, Epoch 600/1000

TRAINWB1, Epoch 800/1000

TRAINWB1, Epoch 1000/1000

TRAINWB1, Maximum epoch reached.

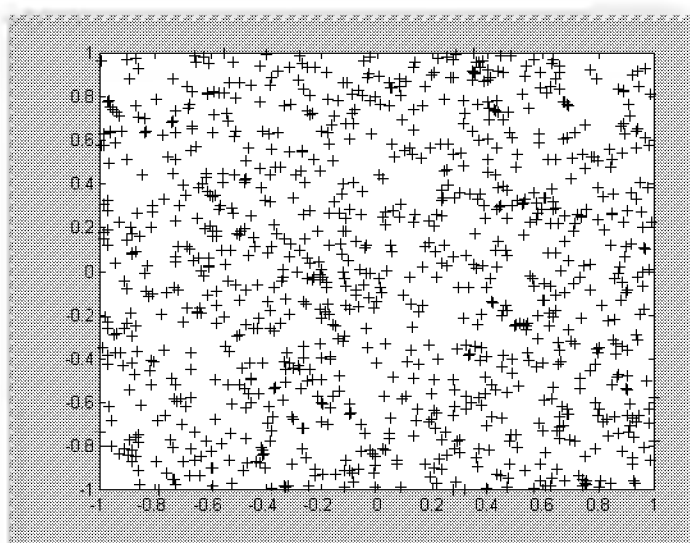


Рис. 4.21. Исходные данные

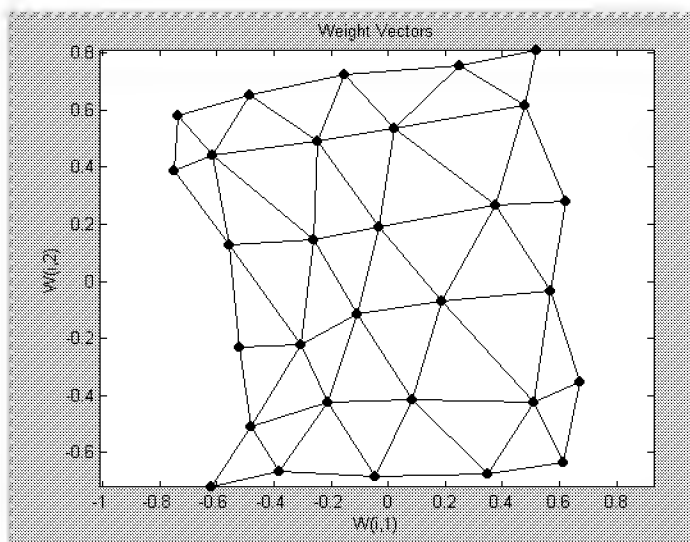


Рис. 4.22. Выявленные центры кластеров

```

» plotsom(net.iw{1,1},net.layers{1}.distances); % Выявленные
центры кластеров
» p = [0.5; 0.3];
» a = sim(net,p) % Опрос сети
a =
    (11,1)    1

```

Предъявленный на этапе тестирования вектор отнесен НС к 11-му классу.

Выявленные центры кластеров представлены на рис. 4.22.

Другие примеры доступны через главное меню MATLAB (пункт Help/Examples and Demos, раздел Toolboxes/Neural Networks).

4.3.10. Использование Simulink при построении нейронных сетей. Пакет Neural Network Toolbox содержит ряд блоков, которые либо могут быть непосредственно использованы для построения нейронных сетей в среде Simulink, либо применяться вместе с рассмотренной выше функцией **gensim**.

Для вызова отмеченной набора блоков, в командной строке необходимо набрать команду **neural**, после выполнения которой появляется окно вида рис. 4.23. Каждый из представленных на

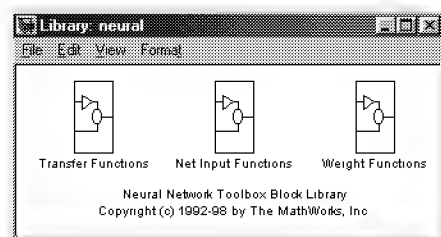


Рис. 4.23. Основные нейросетевые блоки Simulink

рис. 4.23 блоков в свою очередь является набором (библиотекой) некоторых блоков. Рассмотрим их.

Блоки функций активации (**Transfer Functions**). Двойной щелчок левой кнопки мыши на блоке **Transfer Functions** приводит к появлению библиотеки функций активации (рис. 4.24). Каждый из этих блоков данной библиотеки преобразует подаваемый на него вектор в соответствующий вектор той же размерности (табл. 2.1).

Блоки преобразования входов сети. Проводя аналогичную рассмотренной операции, но с блоком **Net Input Functions**, приходим к библиотеке блоков вида рис. 4.25.

Блоки данной библиотеки реализуют рассмотренные выше функции преобразования входов сети.

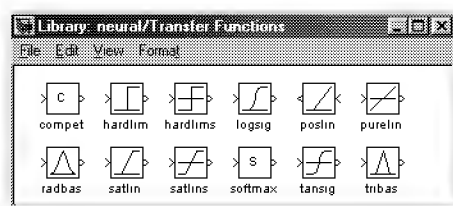


Рис. 4.24. Библиотека функций активации

Блоки весовых коэффициентов. Точно так же (но щелкая левой кнопкой мыши по иконке с надписью Weight Func-

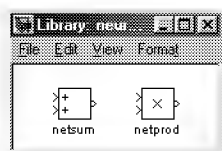


Рис. 4.25. Библиотека блоков преобразования сигналов

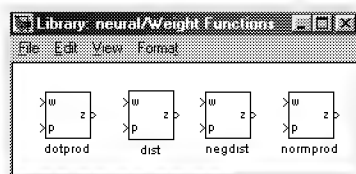


Рис. 4.26. Библиотека блоков весовых коэффициентов

tions) приходим к библиотеке блоков (рис. 4.26), реализующих некоторые функции весов и расстояний.

Отметим, что при задании конкретных числовых значений, при операции со всеми приведенными блоками ввиду особенностей Simulink векторы необходимо представлять как столбцы, а не как строки (как это было до сих пор).

Формирование нейросетевых моделей. Основной функцией для формирования нейросетевых моделей в Simulink является функция **gensim**, записываемая в форме

gensim(net,st),

где **net** — имя созданной НС, **st** — интервал дискретизации (если НС не имеет задержек, ассоциированных с ее входами или слоями, значение данного аргумента устанавливается равным -1).

В качестве примера использования средств Simulink рассмотрим следующий.

Пусть входной и целевой векторы имеют вид

$p = [1 \ 2 \ 3 \ 4 \ 5];$

$t = [1 \ 3 \ 5 \ 7 \ 9].$

Создадим линейную НС и протестируем ее:

```
» p = [1 2 3 4 5];
» t = [1 3 5 7 9];
» net = newlind(p,t);
» y = sim(net,p)
y =
    1.0000    3.0000    5.0000    7.0000    9.0000
```

Затем запустим Simulink командой

```
» gensim(net,-1)
```

Это приведет к открытию блок-диаграммы (рис. 4.27).

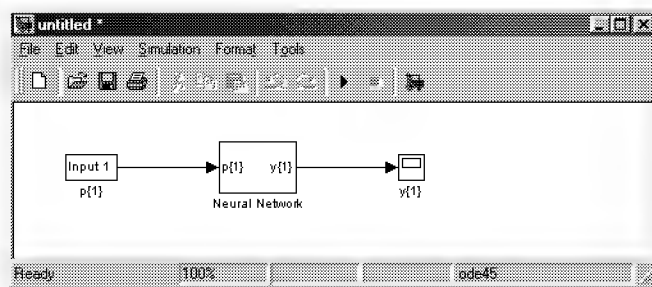


Рис. 4.27. Созданная нейросетевая модель Simulink

Для проведения тестирования модели щелкнем дважды по левой иконке (с надписью Input 1, т.е. Вход 1), что приведет к открытию диалогового окна (рис. 4.28).

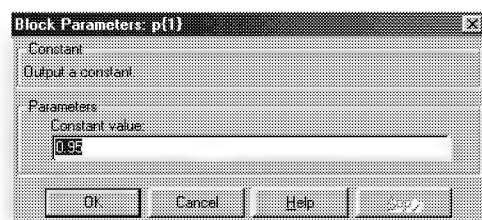


Рис. 4.28. Диалоговое окно задания входа НС

В данном случае блок Input 1 является стандартным блоком задания константы (**Constant**). Изменим значение по умолчанию на 2 и нажмем кнопку **OK**. Затем нажмем кнопку **Start** в меню моделирования. Расчет нового значения сетью производится практически мгновенно. Для его вывода необходимо дважды щелкнуть

мышью по правой иконке (блоку $y(1)$). Результат вычислений отображается рис. 4.29 — он равен 3, как и требуется.

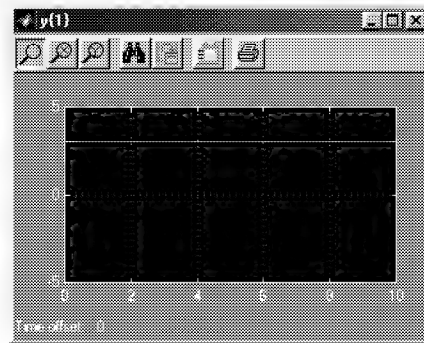


Рис. 4.29. Окно с выходом НС

Отметим, что, дважды щелкая левой кнопкой мыши по блоку Neural Network, затем — по блоку Layer 1, можно получить детальную графическую информацию о структуре сети (рис. 4.30).

С созданной сетью можно проводить различные эксперименты, возможные в среде Simulink; вообще с помощью команды **gensim** осуществляется интеграция созданных нейросетей в блок-диаграм-

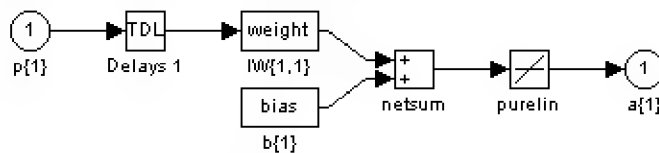


Рис. 4.30. Структура созданной НС

мы этого пакета — с использованием имеющихся при этом инструментов моделирования различных систем (например, встраивание нейросетевого регулятора в систему управления и моделирование последней и т. п.).

Глава 5

ПАКЕТ FUZZY LOGIC TOOLBOX

5.1. Назначение и возможности пакета Fuzzy Logic Toolbox

Пакет Fuzzy Logic Toolbox (пакет нечеткой логики) — это совокупность прикладных программ, относящихся к теории *размытых* или *нечетких* множеств и позволяющих конструировать так называемые нечеткие экспертные и/или управляющие системы.

Основные возможности пакета:

- построение систем нечеткого вывода (экспертных систем, регуляторов, аппроксиматоров зависимостей);
- построение адаптивных нечетких систем (гибридных нейронных сетей);
- интерактивное динамическое моделирование в Simulink.

Пакет позволяет работу:

- в режиме графического интерфейса;
- в режиме командной строки;
- с использованием блоков и примеров пакета Simulink.

5.2. Графический интерфейс Fuzzy Logic Toolbox

5.2.1. Состав графического интерфейса. В состав программных средств Fuzzy Logic Toolbox входят следующие основные программы, позволяющие работать в режиме графического интерфейса:

- редактор нечеткой системы вывода Fuzzy Inference System Editor (FIS Editor или FIS-редактор) вместе со вспомогательными программами — редактором функций принадлежности (Membership Function Editor), редактором правил (Rule Editor), просмотрщиком правил (Rule Viewer) и просмотрщиком поверхности отклика (Surface Viewer);
- редактор гибридных систем (ANFIS Editor, ANFIS-редактор);
- программа нахождения центров кластеров (программа Clustering — кластеризация).

Набор данных программ предоставляет пользователю максимальные удобства для создания, редактирования и использования различных систем нечеткого вывода.

5.2.2. Построение нечеткой аппроксимирующей системы. Командой (функцией) **Fuzzy** из режима командной строки запускается основная интерфейсная программа пакета Fuzzy Logic — редактор нечеткой системы вывода (Fuzzy Inference System Editor,

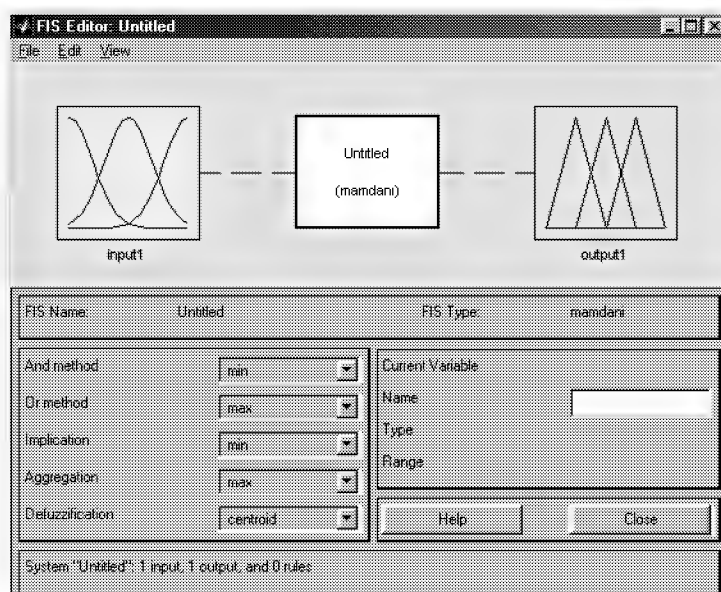


Рис. 5.1. Вид окна FIS Editor

FIS Editor, FIS-редактор). Вид открывающегося при этом окна приведен на рис. 5.1.

Главное меню редактора содержит позиции:

File — работа с файлами моделей (их создание, сохранение, считывание и печать);

Edit — операции редактирования (добавление и исключение входных и выходных переменных);

View — переход к дополнительному инструментарию.

Как говорят англичане, чтобы узнать вкус пудинга, надо его съесть, поэтому представляется целесообразным выяснение различных опций и возможностей данного редактора и связанных с ним других программ изучить на каком-либо конкретном примере.

Попробуем сконструировать нечеткую систему, отображающую зависимость между переменными x и y , заданную с помощью табл. 5.1 (легко видеть, что представленные в таблице данные отображают зависимость $y = x^2$).

Таблица 5.1. Значения x и y

x	1	0.6	0	0.4	1
y	1	0.36	0	0.16	1

Требуемые действия отобразим следующими пунктами.

1. В позиции меню File выбираем опцию New Sugeno FIS (новая система типа Sugeno), при этом в блоке, отображаемом белым квадратом, в верхней части окна редактора появится надпись Untitled2 (sugeno).

2. Щелкнем левой кнопкой мыши по блоку, озаглавленному input1 (вход1). Затем в правой части редактора в поле, озаглавленном Name (Имя), вместо input1 введем обозначение нашего аргумента, т.е. x . Обратим внимание, что если теперь сделать где-нибудь (вне блоков редактора) однократный щелчок мыши, то имя отмеченного блока изменится на x ; то же достигается нажатием после ввода клавиши Enter.

3. Дважды щелкнем по этому блоку. Перед нами откроется окно редактора функций принадлежности Membership Function Editor (см. рис. 5.2). Войдем в позицию меню Edit данного редактора и выберем в нем опцию Add MFs (Add Membership Functions Добавить функций принадлежности). При этом появится диалоговое окно (рис. 5.3), позволяющее задать тип (MF type) и количество (Number of MFs) функций принадлежности (в данном случае все относится к входному сигналу, т.е. к переменной x). Выберем гауссовы функции принадлежности (gaussmf), а их количество зададим равным пяти — по числу значений аргумента в табл. 5.1. Подтвердим ввод информации нажатием кнопки ОК, после чего произойдет возврат к окну редактора функций принадлежности.

4. В поле Range (Диапазон) установим диапазон изменения x от -1 до 1 , т.е. диапазон, соответствующий табл. 5.1. Щелкнем затем левой кнопкой мыши где-нибудь в поле редактора (или нажмем клавишу ввода Enter). Обратим внимание, что после этого произойдет соответствующее изменение диапазона в поле Display Range (Диапазон дисплея).

5. Обратимся к графикам заданных нами функций принадлежности, изображенным в верхней части окна редактора функ-

ций принадлежности. Заметим, что для успешного решения поставленной задачи необходимо, чтобы ординаты максимумов этих

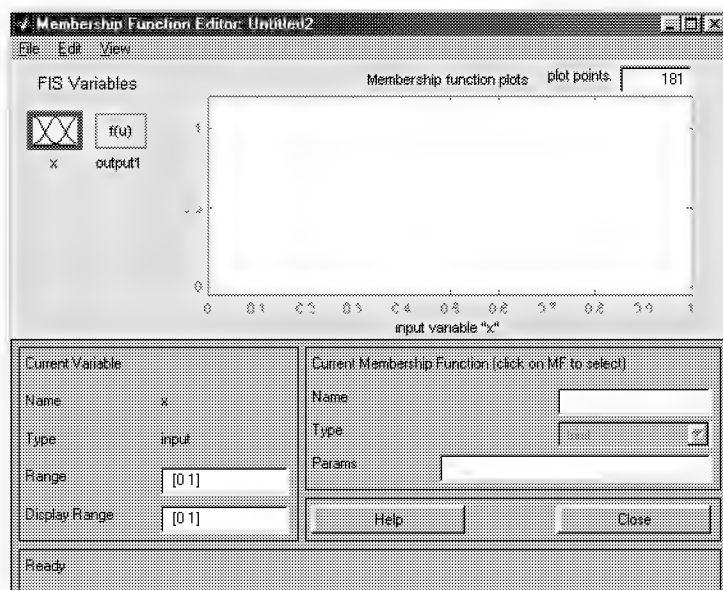


Рис. 5.2. Окно редактора функций принадлежности

функций совпадали с заданными значениями аргумента x . Для левой, центральной и правой функций такое условие выполнено, но две другие необходимо «подвинуть» вдоль оси абсцисс. «Пере-

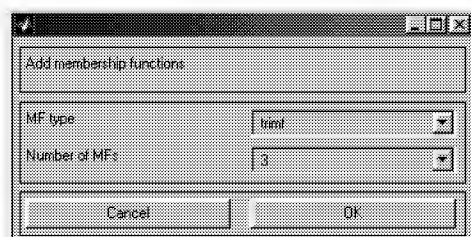


Рис. 5.3. Диалоговое окно задания типа и количества функций принадлежности

движка» делается весьма просто: подводим курсор к нужной кривой и щелкаем левой кнопкой мыши. Кривая выбирается, окрашиваясь в красный цвет, после чего с помощью курсора ее и можно

подвинуть в нужную сторону (более точную установку можно провести, изменяя числовые значения в поле Params (Параметры) в данном случае каждой функции принадлежности соответствуют два параметра, при этом первый определяет размах кривой, а второй — положение ее центра). Для выбранной кривой, кроме этого, в поле Name можно изменять имя (завершая ввод каждого имени нажатием клавиши Enter). Проведем требуемые перемещения кривых и зададим всем пяти кривым новые имена, например:

- самой левой — bp,
- следующей — n,
- центральной — z,
- следующей за ней справа — p,
- самой правой — br.

Нажмем кнопку Close и выйдем из редактора функций принадлежности, возвратившись при этом в окно редактора нечеткой системы (FIS Editor).

6. Сделаем однократный щелчок левой кнопкой мыши по голубому квадрату (блоку), озаглавленному output1 (выход1). В окошке Name заменим имя output1 на y (как в пункте 2).

7. Дважды щелкнем по отмеченному блоку и перейдем к программе — редактору функций принадлежности. В позиции меню Edit выберем опцию Add MFs. Появляющееся диалоговое окно вида рис. 5.3 позволяет задать теперь в качестве функций принадлежности только линейные (linear) или постоянные (constant) — в зависимости от того, какой алгоритм Sugeno (1-го или 0-го порядка) мы выбираем. В рассматриваемой задаче необходимо выбрать постоянные функции принадлежности с общим числом 4 (по числу различных значений y в табл. 5.1). Подтвердим введенные данные нажатием кнопки OK, после чего произойдет возврат в окно редактора функций принадлежности.

8. Обратим внимание, что здесь диапазон (Range) изменения, устанавливаемый по умолчанию — $[0, 1]$, менять не нужно. Изменим лишь имена функций принадлежности (их графики при использовании алгоритма Sugeno для выходных переменных не приводятся), например, задав их как соответствующие числовые значения y , т.е. 0, 0.16, 0.36, 1; одновременно эти же числовые значения введем в поле Params (рис. 5.4). Затем закроем окно нажатием кнопки Close и вернемся в окно FIS-редактора.

9. Дважды щелкнем левой кнопкой мыши по среднему (белому) блоку, при этом раскроется окно еще одной программы редактора правил (Rule Editor). Введем соответствующие правила. При вводе каждого правила необходимо обозначить соот-

ветствие между каждой функцией принадлежности аргумента x и числовым значением y . Кривая, обозначенная нами bn , соответствует $x = -1$, т.е. $y = 1$. Выберем, поэтому в левом поле

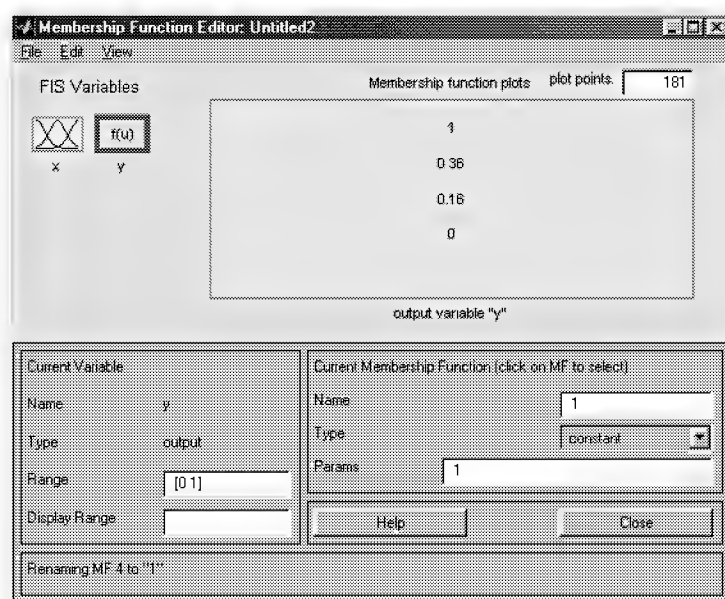


Рис. 5.4. Параметры функций принадлежности переменной y

(с заголовком x is) bn , а в правом 1 и нажмем кнопку Add rule (Добавить правило). Введенное правило появится в окне правил и будет представлять собой запись: 1. If (x is bn) then (y is 1) (1). Аналогично поступим для всех других значений x , в результате чего сформируется набор из 5 правил (см. рис. 5.5). Закроем окно редактора правил и возвратимся в окно FIS-редактора. Построение системы закончено и можно начать эксперименты по ее исследованию. Заметим, что большинство опций выбиралось нами по умолчанию.

10. Предварительно сохраним на диске (используя пункты меню File/Save to disk as...) созданную систему под каким-либо именем, например, Proba.

11. Выберем позицию меню View. Как видно из выпадающего при этом подменю, с помощью пунктов Edit membership functions и Edit rules можно совершить переход к двум выше рассмотренным программам — редакторам функций принадлежности и пра-

вил (то же можно сделать и нажатием клавиш Ctrl+2 или Ctrl+3), но сейчас нас будут интересовать два других пункта — View rules

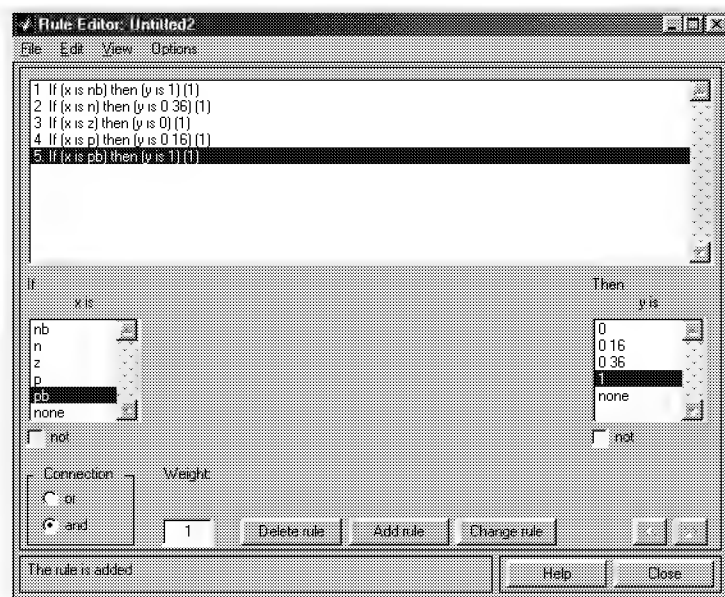


Рис. 5.5. Окно редактора правил

(Просмотр правил) и View surface (Просмотр поверхности). Выберем пункт View rules, при этом откроется окно (см. рис. 5.6) еще одной программы — просмотра правил (Rule Viewer).

12. В правой части окна в графической форме представлены функции принадлежности аргумента x , в левой — переменной выхода y с пояснением механизма принятия решения. Красная вертикальная черта, пересекающая графики в правой части окна, которую можно перемещать с помощью курсора, позволяет изменять значения переменной входа (это же можно делать, задавая числовые значения в поле Input (Вход)), при этом соответственно изменяются значения y в правой верхней части окна. Зададим, например, $x = 0.5$ в поле Input и нажмем затем клавишу ввода (Enter). Значение y сразу изменится и станет равным 0.202. Таким образом, с помощью построенной модели и окна просмотра правил можно решать задачу интерполяции, т.е. задачу, решение которой и требовалось найти. Изменение аргумента путем

перемещения красной вертикальной линии очень наглядно демонстрирует, как система определяет значения выхода.

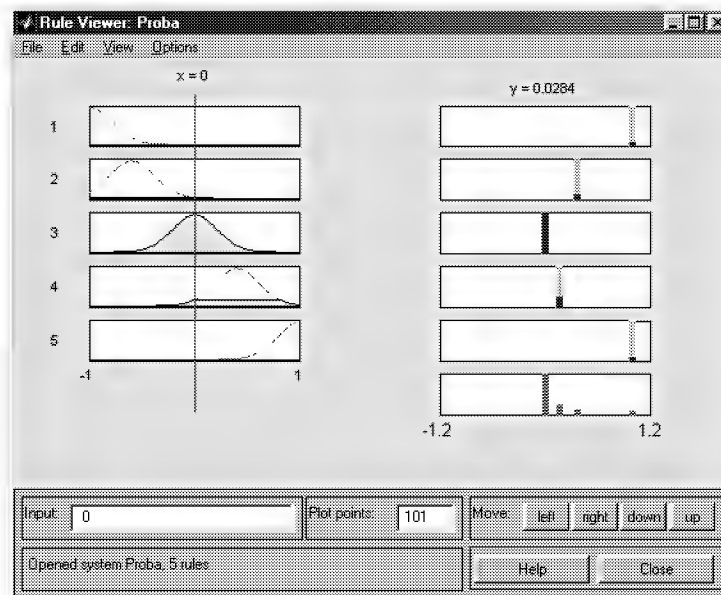


Рис. 5.6. Окно просмотра правил

13. Закроем окно просмотра правил и выбором пункта меню View/View surface перейдем к окну просмотра поверхности отклика (выхода), в нашем случае — к просмотру кривой $y(x)$ (см. рис. 5.7). Видно, что смоделированное системой по таблице данных (табл. 5.1) отображение не очень-то напоминает функцию x^2 . Ну что ж, ничего удивительного в этом нет: число экспериментальных точек невелико, да и параметры функций принадлежности (для x) выбраны, скорее всего, неоптимальным образом. Ниже мы рассмотрим возможность улучшения качества подобной модели.

В заключение рассмотрения примера отметим, что с помощью вышеуказанных программ-редакторов на любом этапе проектирования нечеткой модели в нее можно внести необходимые коррективы, вплоть до задания какой-либо особенной пользовательской функции принадлежности. Из опций, устанавливаемых в FIS-редакторе по умолчанию при использовании алгоритма Sugeno, можно отметить:

- логический вывод организуется с помощью операции умножения (prod);
- композиция — с помощью операции логической суммы (вероятностного ИЛИ, probor);
- приведение к четкости — дискретным вариантом центроидного метода (взвешенным средним, wtaver).

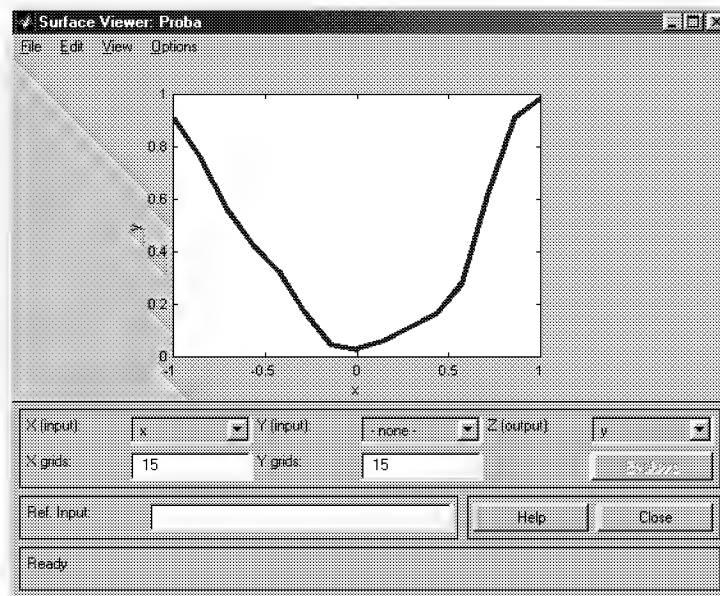


Рис. 5.7. Окно просмотра поверхности отклика

Используя соответствующие поля в левой нижней части окна FIS-редактора, данные опции можно, при желании, изменить.

5.2.3. Построение экспертной системы: сколько дать «на чай»?

Рассмотрим теперь методику построения нечеткой экспертной системы, которая должна помочь пользователю с ответом на вопрос: сколько дать «на чай» официанту за обслуживание в ресторане? (Предположим, речь идет о местах, где такие чаевые принято давать, например, в ресторанах Парижа или Рио-де-Жанейро.)

Основываясь на каких-то устоявшихся обычаях и интуитивных представлениях, примем, что задача о чаевых может быть описана следующими предложениями.

1. Если обслуживание плохое или еда подгоревшая, то чаевые — малые.

2. Если обслуживание хорошее, то чаевые — средние.
3. Если обслуживание отличное или еда превосходная, то чаевые — щедрые.

Качество обслуживания и еды будем оценивать по 10-балльной системе (0 — наихудшая оценка, 10 — наилучшая).

Будем предполагать, далее, что малые чаевые составляют около 5% от стоимости обеда, средние — около 15% и щедрые — примерно 25%.

Заметим, что представленной информации, в принципе, достаточно для проектирования нечеткой экспертной системы. Такая система будет иметь 2 входа (которые условно можно назвать «сервис» и «еда»), один выход («чаевые»), три правила типа «если... то» (в соответствии с тремя приведенными предложениями) и по три значения (соответственно, 0 баллов, 5 баллов, 10 баллов и 5%, 15%, 25%) для центров функций принадлежности входов и выхода. Построим данную систему, используя алгоритм вывода Mamdani и, как в предыдущем примере, описывая требуемые действия по пунктам.

1. Командой **fuzzy** запускаем FIS-редактор. По умолчанию, исходный алгоритм вывода — типа Mamdani (о чем говорит надпись в центральном белом блоке) и здесь никаких изменений не требуется, но в системе должно быть два входа, поэтому через пункт меню Edit/Add input добавляем в систему этот второй вход (в окне редактора появляется второй желтый блок с именем input2). Делая далее однократный щелчок левой кнопкой мыши по блоку input1, меняем в поле имени его имя на «сервис», завершая ввод нового имени нажатием клавиши Enter. Аналогичным образом устанавливаем имя «еда» блоку input2 и «чаевые» — выходному блоку (справа сверху) output1. Присвоим сразу же и имя всей системе, например, «tip» (по-английски это и есть чаевые), выполнив это через пункт меню File/Save to workspace as... (Сохранить в рабочем пространстве как...). Вид окна редактора после указанных действий приведен на рис. 5.8.

2. Зададим теперь функции принадлежности переменных. Напомним еще раз, что программу-редактор функций принадлежности можно открыть тремя способами:

- через пункт меню View/Edit membership functions...,
- двойным щелчком левой кнопки мыши по иконке, отображающей соответствующую переменную,
- нажатием клавиш Ctrl+2.

Любым из приведенных способов перейдем к данной программе.

Задание и редактирование функций принадлежности начнем с переменной «сервис». Сначала в полях Range и Display Range

установим диапазон изменения и отображения этой переменной от 0 до 10 (баллов), подтверждая ввод нажатием клавиши Enter. Затем через пункт меню Edit/Add MFs перейдем к диалоговому окну вида рис. 5.3 и зададим в нем функции принадлежности гауссова типа (gaussmf) с общим числом 3. Нажмем кнопку OK и возвратимся в окно редактора функций принадлежности. Не изменяя размах и положение заданных функций, заменим только их имена на «плохой», «хороший» и «отличный» (как в пункте 5 предыдущего примера).

Щелчком левой кнопки мыши по иконке «еда» войдем в окно редактирования функций принадлежности для этой переменной. Зададим сначала диапазон ее изменения от 0 до 10, а затем, по-

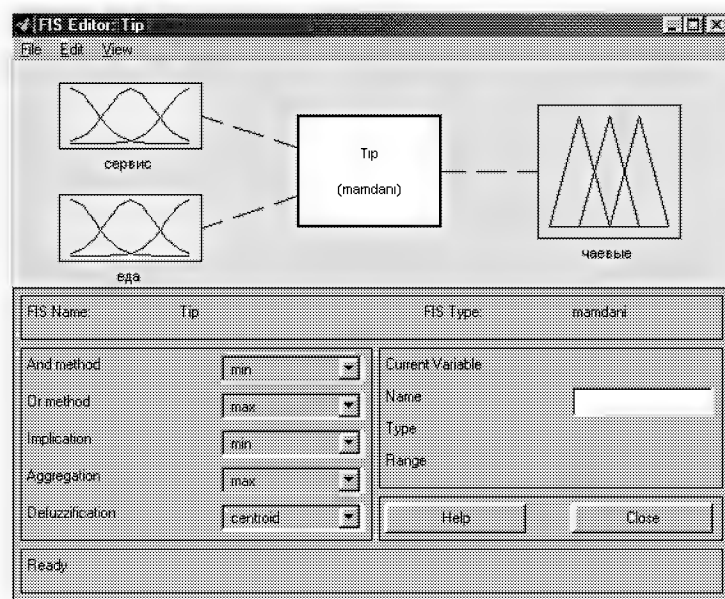


Рис. 5.8. Вид окна FIS-редактора после задания структуры системы

ступая как ранее, зададим две функции принадлежности трапецидальной формы с параметрами, соответственно, $[0 \ 0 \ 1 \ 3]$ и $[7 \ 9 \ 10 \ 10]$ и именами «подгоревшая» и «превосходная».

Для выходной переменной «чайевые» укажем сначала диапазон изменения — от 0 до 30, потом зададим три функции принадлежности треугольной формы с именами «малые», «средние», «щедрые» так, как это представлено на рис. 5.9. Заметим, что можно,

разумеется, задать и какие-либо другие функции или выбрать их другие параметры.

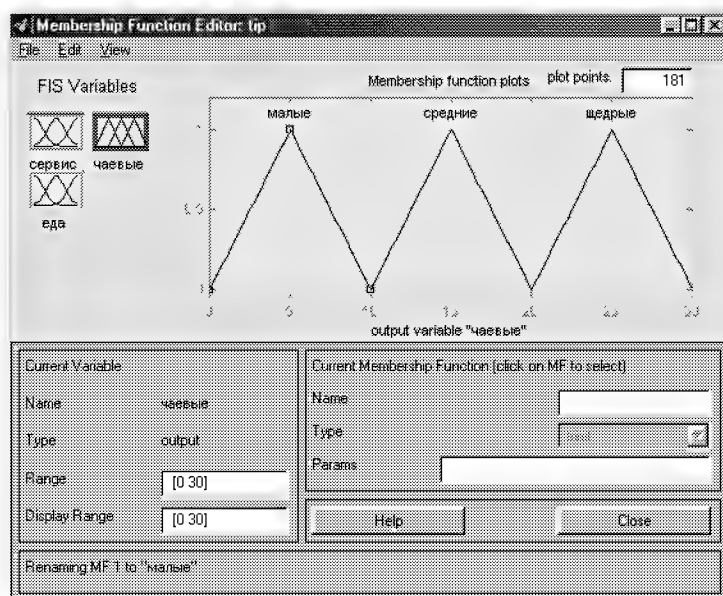


Рис. 5.9. Функции принадлежности переменной «чайевые»

3. Перейдем к конструированию правил. Для этого выберем пункт меню View/Edit rules... Далее ввод правил производится так же, как в п. 9 предыдущего примера, и в соответствии с предложениями, описывающими задачу. Заметим, что в первом и третьем правилах в качестве «связки» в предпосылках правила необходимо использовать не «И» (and), а «ИЛИ» (or); при вводе второго правила, где отсутствует переменная «еда», для нее выбирается опция none. Итоговый набор правил отображен рис. 5.10 и выглядит следующим образом:

1. If (сервис is плохой) or (еда is подгоревшая) then (чайевые is малые) (1)
2. If (сервис is хороший) then (чайевые is средние) (1)
3. If (сервис is отличный) или (еда is превосходная) then (чайевые is щедрые) (1)

Такая (подробная, verbose) запись представляется достаточно понятной; единица в скобках после каждого правила указывает его «вес» (Weight), т.е. значимость правила. Данный вес можно

менять, используя соответствующее поле в левой нижней части окна редактора правил. Правила представимы и в других формах: символической (symbolic) и индексной (indexed), при этом пере-

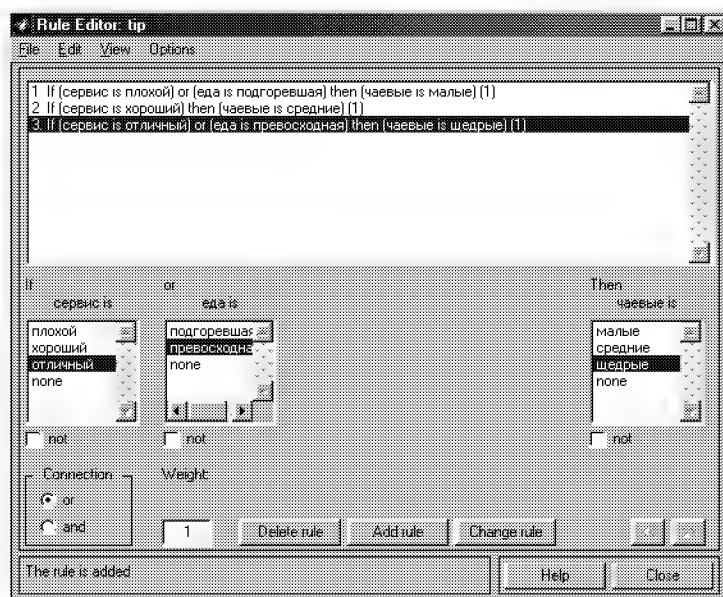


Рис. 5.10. Итоговый набор правил в задаче о чаевых

ход от одной формы к другой происходит через опции пункта меню редактора правил Options/Format. Вот как выглядят рассмотренные правила в символической форме:

1. (сервис==плохой)|(еда==подгоревшая)⇒
⇒(чаевые=малые) (1)
2. (сервис==хороший)⇒(чаевые=средние) (1)
3. (сервис==отличный)|(еда==превосходная)⇒
⇒(чаевые=щедрые) (1)

По-видимому, здесь тоже понятно все.

Наконец, самый сжатый формат представления правил — индексный — является тем форматом, который в действительности используется программой. В этом формате приведенные правила выглядят так:

- 1 1, 1 (1): 2
- 2 0, 2 (1): 2
- 3 2, 3 (1): 2

Здесь первая колонка относится к первой входной переменной (соответственно, первое, второе или третье возможное значение), вторая — ко второй, третья (после запятой) — к выходной переменной, цифра в скобках показывает вес правила и последняя цифра (после двоеточия) — на тип «связки» (1 для «И», 2 для «ИЛИ»).

На этом, собственно, конструирование экспертной системы закончено. Сохраним ее на диске под выбранным именем (tip).

4. Самое время теперь проверить систему в действии. Откроем (через пункт меню View/View rules...) окно просмотра правил и установим значения переменных: сервис=0 (т.е. никуда не годный), еда=10 (т.е. превосходная). Увидим ответ: чаевые=15 (т.е. средние). Ну что ж, с системой не поспоришь, надо платить (рис. 5.11). Можно проверить и другие варианты. В частности (может быть, не без удивления), выяснится, что нашей системой обслуживание ценится больше, чем качество еды: при наборе

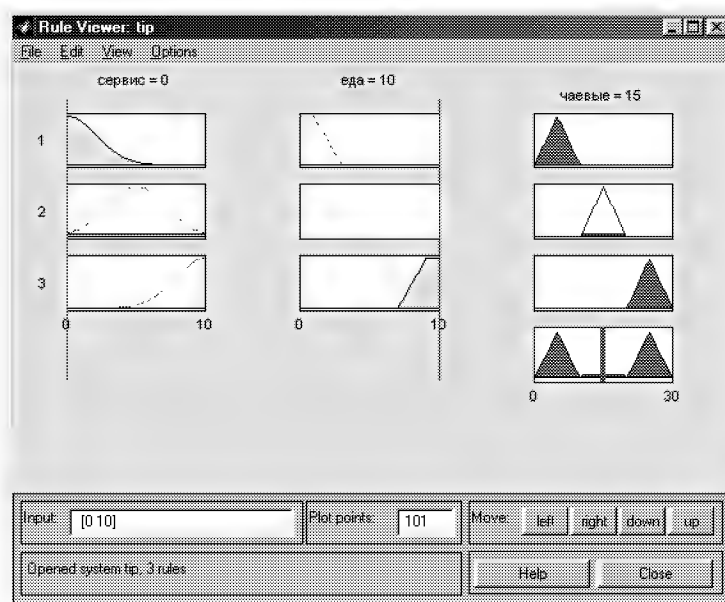


Рис. 5.11. Окно просмотра правил в задаче о чаевых

«сервис=10, еда=3» система советует определить размер чаевых в 23.9%, в то время как набору «сервис=3, еда=10» размер чаевых по рекомендации системы — 16.6% (от стоимости обеда). Впро-

чем, ничего удивительного здесь нет: это мы сами (не особенно подозревая об этом) заложили в систему соответствующие знания в виде совокупности приведенных правил.

Подтверждением отмеченной зависимости выходной переменной от входных может служить вид поверхности отклика, который представляется при выборе пункта меню View/View surface

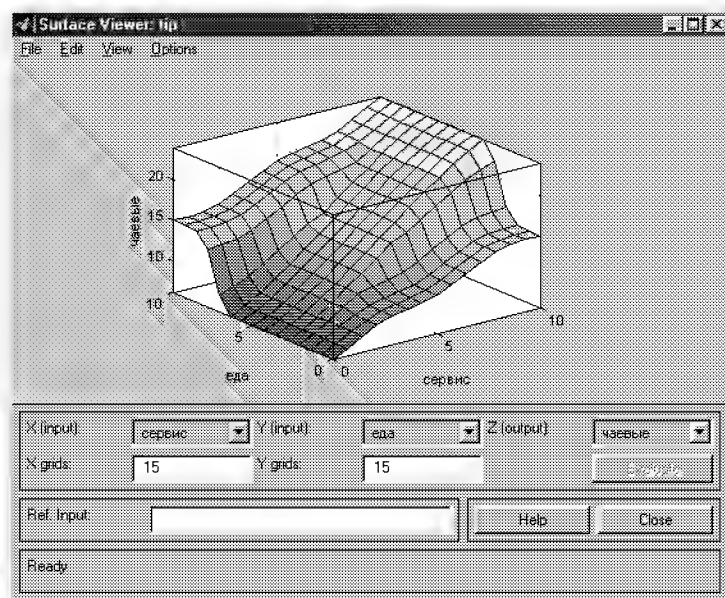


Рис. 5.12. Графический вид зависимости выходной переменной от входных

(рис. 5.12); обратите внимание, что с помощью мышки график можно поворачивать во все стороны.

В открывшемся окне, меняя имена переменных в полях ввода (X(input) и Y(input)), можно задать и просмотр одномерных зависимостей, например «чайевых» от «еды» (рис. 5.13).

5.2.4. Экспорт и импорт результатов. Когда вы сохраняете созданную вами нечеткую систему, используя пункты меню File/Save to disk или File/Save to disk as..., на диске создается текстовый (ASCII) файл достаточно простого формата с расширением .fis, который можно просматривать, при необходимости редактировать вне системы MATLAB, а также использовать повторно при последующих сеансах работы с системой. Однако сохранение с

использованием пунктов File/Save to workspace или File/Save to workspace as... на самом деле только «легализует» созданную вами систему (под каким-либо именем) в среде MATLAB в течение те-

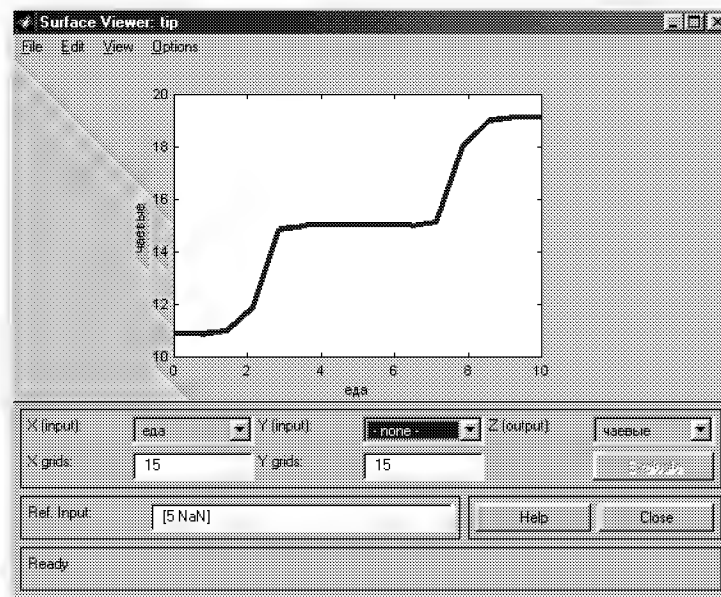


Рис. 5.13. Одномерная зависимость размера чаевых от качества еды

кущего сеанса работы и не допускает ее повторного использования в других сеансах.

5.2.5. Создание пользовательских функций принадлежности.

Если по каким-либо причинам вас не устраивает ни одна из встроенных функций принадлежности, вы можете создать и использовать собственную подходящую функцию. Такая функция должна быть создана как M-файл со значениями от 0 до 1 и с числом аргументов не более 16. Приведем этапы создания данной функции под некоторым именем *custmf*.

1. Создается соответствующий M-файл с именем *custmf.m*.

2. Выбирается пункт Edit/Add custom MF (Редактирование/Добавить пользовательскую функцию принадлежности) в меню редактора функций принадлежности.

3. В поле M-File function name появляющегося диалогового окна Add customized membership function вводится имя созданного M-файла (*custmf*).

4. В поле Parameter list данного окна вводятся необходимые числовые параметры.

5. Наконец, в поле MF name (Имя функции принадлежности) вводится какое-либо (уникальное) имя задаваемой функции (например, custmf).

6. Указанный ввод подтверждается нажатием кнопки ОК (см. рис. 5.14).

Ниже приведен пример М-файла некоторой пользовательской функции принадлежности дискретного типа, имеющей имя testmfl

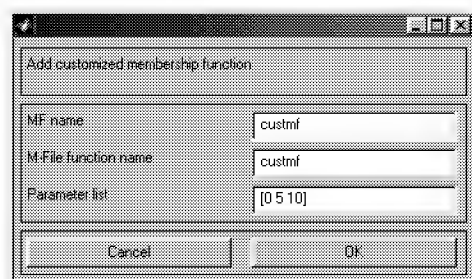


Рис. 5.14. Окно задания функции принадлежности пользователя

и зависящей от 8 числовых параметров (каждый из диапазона $[0, 10]$):

```
function out = testmfl(x, params)
for i = 1:length(x)
if x(i) < params(1)
y(i) = params(1);
elseif x(i) < params(2)
y(i) = params(2);
elseif x(i) < params(3)
y(i) = params(3);
elseif x(i) < params(4)
y(i) = params(4);
elseif x(i) < params(5)
y(i) = params(5);
elseif x(i) < params(6)
y(i) = params(6);
elseif x(i) < params(7)
y(i) = params(7);
elseif x(i) < params(8)
y(i) = params(8);
else
```



```
y(i) = 0;  
end  
end  
out = .1*y'
```

5.3. Графический интерфейс гибридных систем

Графический интерфейс гибридных (нечетких) нейронных систем вызывается функцией (из режима командной строки) **anfisedit**. Исполнение функции приводит к появлению окна редактора гибридных систем (ANFIS Editor, ANFIS-редактор), вид которого приведен на рис. 5.15.

С помощью данного редактора осуществляется создание или загрузка структуры гибридной системы, просмотр структуры,

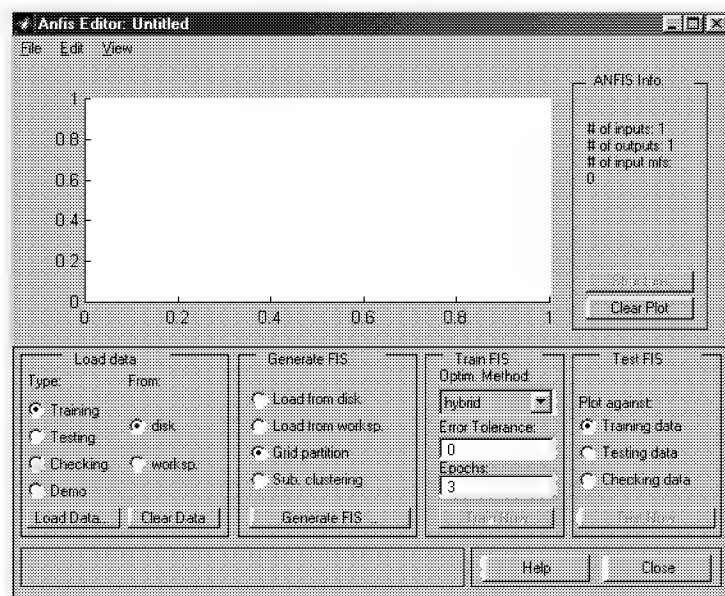


Рис. 5.15. Окно редактора гибридных систем

настройка ее параметров, проверка качества функционирования такой системы. Создание структуры и настройка параметров и проверка осуществляются по выборкам (наборам данных) — обучающей (Training data), проверочной (Checking data) и тестирующей (Testing data), которые предварительно должны быть представлены в виде текстовых файлов (с расширением .dat и разделен-

телями-табуляциями), первые колонки которых соответствуют входным переменным, а последняя (левая) — единственной выходной переменной; количество строк в таких файлах равно количеству образцов (примеров). Так, обучающая выборка, сформированная по табл. 5.1, представляется в виде

-1	1
-0.6	0.36
0.0	0.00
0.4	0.16
1	1

Строгих рекомендаций по объемам указанных выборок не существует, по-видимому, лучше всего исходить из принципа «чем больше, тем лучше». Обучающая и проверочная выборки непосредственно задействуются в процессе настройки параметров гибридной сети (проверочная — для выяснения ситуации: нет ли так называемого переобучения сети, при котором ошибка для обучающей последовательности стремится к нулю, а для проверочной возрастает; впрочем, наличие проверочной выборки не является строго необходимым, оно лишь крайне желательно). Тестовая (или тестирующая выборка) применяется для проверки качества функционирования настроенной (обученной) сети.

Поясним пункты меню и опции редактора.

Пункты меню File и View, в общем идентичны аналогичным пунктам FIS-редактора, за тем исключением, что здесь работа может происходить только с алгоритмом нечеткого вывода Sugeno. Пункт меню Edit содержит единственный подпункт — Undo (Отменить выполненное действие).

Набор опций Load data (Загрузить данные) в нижней левой части окна редактора включает в себя:

- тип (Type) загружаемых данных (для обучения — Training, для тестирования — Testing, для проверки — Checking, демонстрационные — Demo);
- место, откуда должны загружаться данные (с диска — disk или из рабочей области MATLAB-workspace).

К данным опциям относятся две кнопки, нажатие на которых приводит к требуемым действиям — Load Data... (Загрузить данные) и Clear Data (очистить, т.е. стереть введенные данные).

Следующая группа опций (в середине нижней части окна ANFIS-редактора) объединена под именем Generate FIS (Создание нечеткой системы вывода). Данная группа включает в себя опции:

- загрузку структуры системы с диска (Load from disk);

- загрузку структуры системы из рабочей области MATLAB (Load from worksp.);
- разбиение (деление) областей определения входных переменных (аргументов) на подобласти — независимо для каждого аргумента (Grid partition);
- разбиение всей области определения аргументов (входных переменных) на подобласти — в комплексе для всех аргументов (Subtract clustering или Sub. clustering), а также кнопку Generate FIS, нажатие которой приводит к процессу создания гибридной системы с точностью до ряда параметров.

Следующая группа опций — Train FIS (Обучение нечеткой системы вывода) — позволяет определить метод «обучения» (Optim. Method) системы (т.е. метод настройки ее параметров) — гибридный (hybrid) или обратного распространения ошибки (backprop), установить уровень текущей суммарной (по всем образцам) ошибки обучения (Error Tolerance), при достижении которого процесс обучения заканчивается и количество циклов обучения (Epochs), т.е. количество «прогонов» всех образцов (или примеров) обучающей выборки; процесс обучения, таким образом заканчивается либо при достижении отмеченного уровня ошибки обучения, либо при проведении заданного количества циклов.

Кнопка Train Now (Начать обучение) процесс обучения, т.е. процесс настройки параметров гибридной сети.

В правом верхнем углу окна ANFIS-редактора выдается информация (ANFIS Info.) о проектируемой системе: о количестве входов, выходов, функций принадлежности входов; нажатие кнопки Structure (Структура) позволяет увидеть структуру сети. Кнопка Clear (Очистить) позволяет стереть все результаты.

Опции Test FIS в правом нижнем углу окна позволяют провести проверку и тестирование созданной и обученной системы с выводом результатов в виде графиков (соответствующие графики для обучающей выборки Training data, тестирующей выборки Testing data и проверочной выборки Checking data. Кнопка Test Now позволяет запустить указанные процессы.

Работу с редактором рассмотрим на примере восстановления зависимости $y = x^2$ по данным табл. 5.1. Предположим, что эти данные сохранены в файле Proba.dat. Создание и проверку системы, как и раньше, проведем по этапам.

1. В окне ANFIS-редактора выберем тип загружаемых данных Training и нажмем кнопку Load data. В последующем стандартном окне диалога укажем местоположение и имя файла. Его открытие приводит к появлению в графической части окна редактора набора точек, соответствующих введенным данным (рис. 5.16).

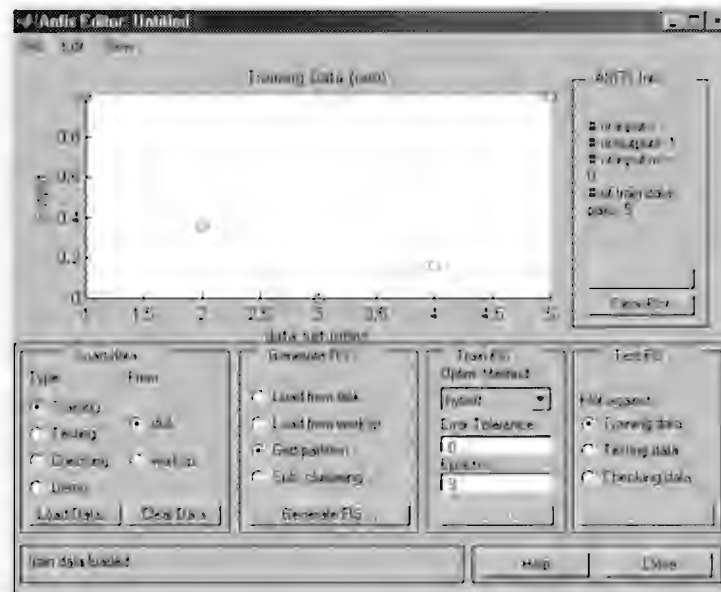


Рис. 5.16. Окно ANFIS-редактора после загрузки обучающей выборки

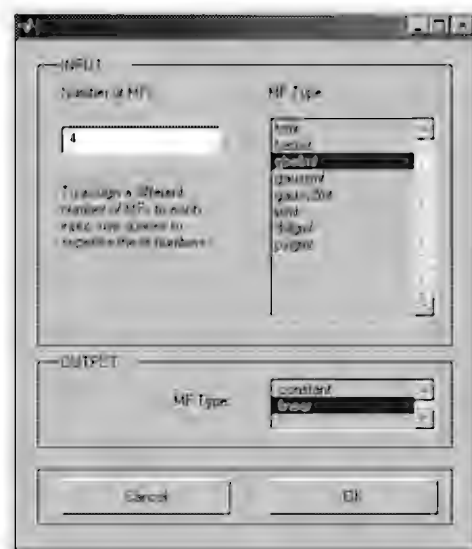


Рис. 5.17. Окно задания функций принадлежности

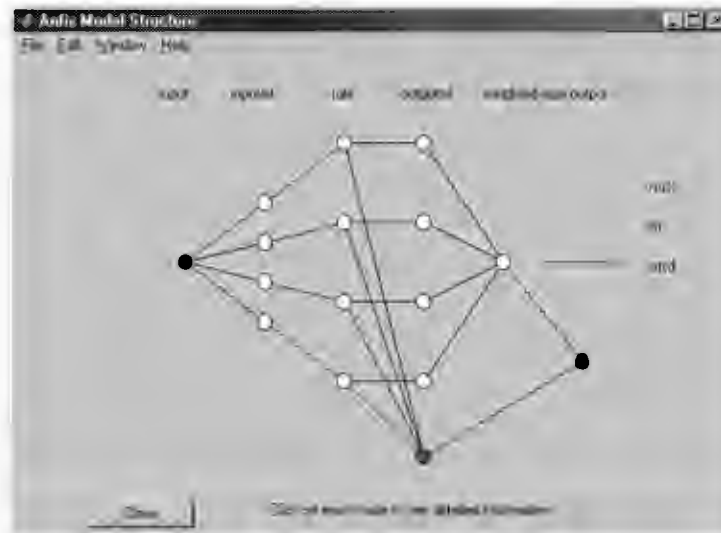


Рис. 5.18. Структура созданной гибридной сети

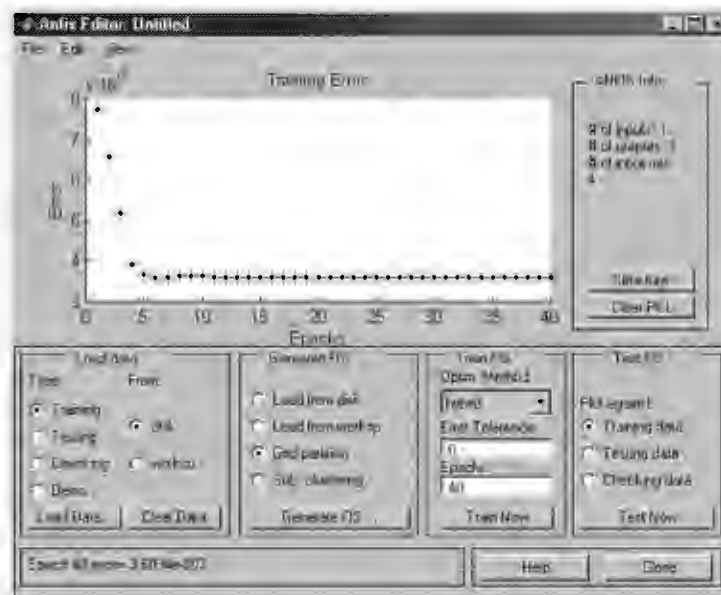


Рис. 5.19. Результат обучения сети

2. В группе опций Generate FIS по умолчанию активизирована опция Grid partition. Не будем ее изменять и нажмем кнопку Generate FIS, после чего появится диалоговое окно (рис. 5.17) для задания числа и типов функций принадлежности. Сохраним все установки по умолчанию, согласившись с ними нажатием кнопки ОК. Произойдет возврат в основное окно ANFIS-редактора. Теперь структура гибридной сети создана, и ее графический вид можно просмотреть с помощью кнопки Structure (рис. 5.18).

3. Перейдем к опциям Train FIS. Не будем менять задаваемые по умолчанию метод настройки параметров (hybrid — гибридный) и уровень ошибки (0), но количество циклов обучения изменим на 40, после чего нажмем кнопку начала процесса обучения (Train Now). Получившийся результат в виде графика ошибки сети в зависимости от числа проведенных циклов обучения (из которого следует, что фактически обучение закончилось после пятого цикла) представлен на рис. 5.19.

4. Теперь нажатием кнопки Test Now можно начать процесс тестирования обученной сети, но, поскольку использовалась только одна обучающая выборка, ничего особенно интересного ожидать не приходится. Действительно, выход обученной системы практически совпадает с точками обучающей выборки (рис. 5.20).

5. Сохраним разработанную систему в файл на диске с именем Probal (с расширением .fis) и для исследования разработанной системы средствами FIS-редактора из командной строки MATLAB выполним команду **fuzzy**, а затем через пункты меню File/Open FIS from disk... откроем созданный файл. С созданной системой можно теперь выполнять все приемы редактирования (изменение имен переменных и т. п.) и исследования, которые были рассмотрены выше. Здесь нетрудно, кстати, убедиться, что качество аппроксимации данных существенно не улучшилось — слишком мало данных.

Что можно сказать про эффективность использования гибридных систем (и ANFIS-редактора)?

В данном случае используется только один алгоритм нечеткого вывода — Sugeno (нулевого или первого порядков), может быть только одна выходная переменная, всем правилам приписывается один и тот же единичный вес. Вообще говоря, возникают значительные проблемы при большом (более 5–6) количестве входных переменных. Это — ограничения и недостатки подхода.

Его несомненные достоинства: практически полная автоматизация процесса создания нечеткой (гибридной) системы, возможность просмотра сформированных правил и придания им содержа-

тельной (лингвистической) интерпретации, что позволяет, кстати говоря, рассматривать аппарат гибридных сетей как средство извлечения знаний из баз данных и существенно отличает данные сети от классических нейронных.

Рекомендуемая область применения: построение аппроксиматоров зависимостей по экспериментальным данным, построение

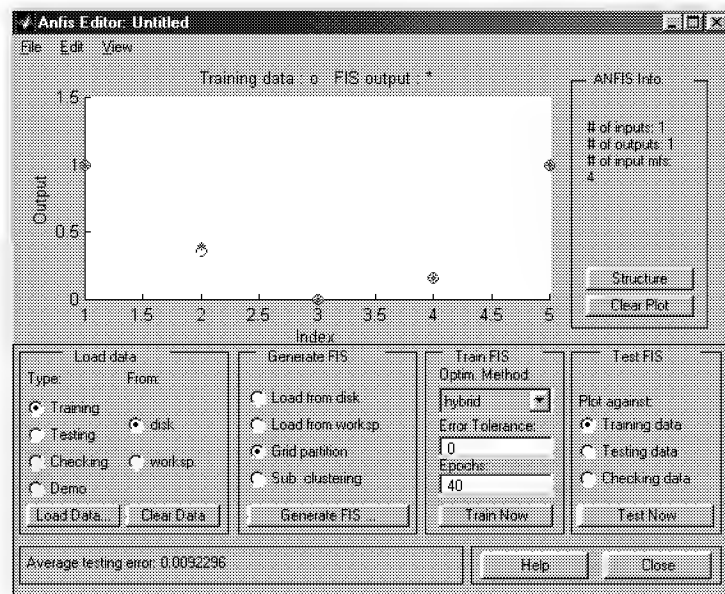


Рис. 5.20. Результат тестирования обученной системы

систем классификации (в случае бинарной или дискретной выходной переменной), изучение механизма явлений.

5.4. Графический интерфейс программы кластеризации

В пакет Fuzzy Logic Toolbox входит еще одна программа, позволяющая работу в режиме графического интерфейса, — программа Clustering (Кластеризация) выявления центров кластеров, т.е. точек в многомерном пространстве данных, около которых группируются (скапливаются) экспериментальные данные. Выявление подобных центров, надо сказать, является значимым этапом при предварительной обработке данных, поскольку позволяет сопоставить с этими центрами функции принадлежности переменных при последующем проектировании системы нечеткого вывода.

Запуск программы Clustering осуществляется командой (функцией) **findcluster**. В появляющемся окне программы имеется (вверху) главное меню, содержащее достаточно стандартный набор пунктов (File, Edit, Window, Help) и набор управляющих кнопок и опций (справа). К этим кнопкам относятся:

- кнопка загрузки файла данных Load Data,
- кнопка выбора алгоритма кластеризации Method,
- четыре расположенные ниже кнопки опций алгоритма (их названия меняются в зависимости от выбранного алгоритма),
- кнопка начала итеративного процесса нахождения центров кластеров (кластеризации) — Start,
- кнопка сохранения результатов кластеризации (Save Center),
- кнопка очистки (стирания) графиков (Clear Plot),
- кнопка справочной информации (Info),
- кнопка завершения работы с программой (Close).

В программе используются два алгоритма выявления центров кластеров: Fuzzy c-means (который можно перевести как «Алгоритм нечетких центров») и Subtractive clustering («Вычитающая

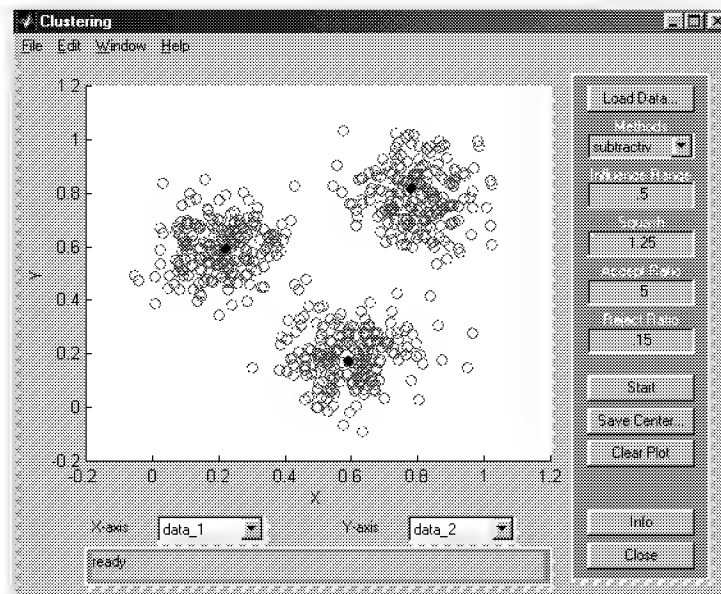


Рис. 5.21. Результат работы программы Clustering (центры кластеров окрашены в черный цвет)

кластеризация»). Если не вдаваться в их детальное теоретическое изложение, а ограничиться выявлением различий на уровне пользователя, то можно отметить, что алгоритм Fuzzy c-means, являясь, пожалуй, более точным (если понятие точности вообще здесь применимо), для своей работы требует задания таких опций, как число кластеров (кнопка Cluster num.) и число итераций (кнопка Max Iteration#). Ну, если число итераций еще можно задать как-то наугад, то ошибка в задании числа кластеров может привести к неприятным последствиям. Алгоритм Subtractive clustering менее точен, но и менее требователен к априорной информации; при работе с ним можно сохранить опции, заданные в программе по умолчанию. На рис. 5.21 приведен пример использования программы для фала данных clusterdemo.dat из директории Matlab/toolbox/fuzzy/fuzdemos/ при использовании алгоритма Subtractive clustering. Заметим, что выводится только двумерное поле рассеяния, но изменяя переменные в соответствующих полях (X-axis и Y-axis), можно «просмотреть» все многомерное пространство переменных.

5.5. Работа с Fuzzy Logic Toolbox в режиме командной строки

5.5.1. Возможности работы в режиме командной строки. Пакет Fuzzy Logic располагает большим набором функций, исполняемых из командной строки MATLAB и позволяющих, в принципе, не использовать при работе с системами нечеткого вывода рассмотренные программы графического интерфейса. Все функции делятся на группы:

- 1) вызова программ графического интерфейса;
- 2) задания функций принадлежности;
- 3) создания, редактирования, просмотра, открытия и сохранения систем нечеткого вывода;
- 4) дополнительные;
- 5) различные;
- 6) вызова диалоговых окон интерфейса;
- 7) блоков Simulink;
- 8) демонстрации возможностей пакета.

5.5.2. Функции вызова программ графического интерфейса. К этой группе относятся функции:

fuzzy — вызов FIS-редактора;
mfedit — вызов редактора функций принадлежности;
ruleedit — вызов редактора правил;
ruleview — вызов программы просмотра правил;

surfview — вызов программы просмотра поверхности отклика;
anfisedit — вызов ANFIS-редактора (только для систем, использующих алгоритм Sugeno и имеющих одну выходную переменную);

findcluster — вызов программы кластеризации.

Использование первых шести функций с аргументом (например, **fuzzy(a)**), где **a** — имя переменной рабочего пространства, присвоенное системе нечеткого вывода), открывает соответствующую программу с одновременной загрузкой в нее рассматриваемой системы.

Функция **findcluster(имя_файла)** открывает программу кластеризации с одновременной загрузкой указанного файла данных.

5.5.3. Задание функций принадлежности. В данную группу включены 11 функций (по числу функций принадлежности, используемых в пакете Fuzzy Logic).

1. Функция **dsigmf**.

Запись: **y = dsigmf(x,[a1 c1 a2 c2])**

Описание. Задается функция принадлежности, определяемая как разность двух сигмоидальных функций. Сигмоидальная функция, как известно, описывается выражением

$$f(x, a, c) = \frac{1}{1 + \exp(-a(x - c))}$$

и зависит от двух числовых параметров a и c . Описываемая функция, как отмечено, является разностью двух сигмоидальных:

$$f_1(x, a_1, c_1) - f_2(x, a_2, c_2),$$

и зависит от четырех параметров a_1, c_1, a_2, c_2 (вектора параметров **[a1 c1 a2 c2]**).

Пример

```
» x = 0:0.1:10;
» y = dsigmf(x,[5 2 5 7]);
» plot(x,y)
» xlabel('dsigmf, P = [5 2 5 7]')
```

2. Функция **gauss2mf**.

Запись: **y = gauss2mf(x,[sig1 c1 sig2 c2])**

Описание. Задается функция принадлежности, являющаяся разностью двух гауссовых функций, определяемая соотношением

$$f(x, \sigma_1, c_1, \sigma_2, c_2) = \exp(-(x - c_1)^2 / \sigma_1^2) - \exp(-(x - c_2)^2 / \sigma_2^2)$$

и зависящую от четырех параметров $(\sigma_1, c_1, \sigma_2, c_2)$ или вектора параметров $[\text{sig1 } c1 \text{ sig2 } c2]$.

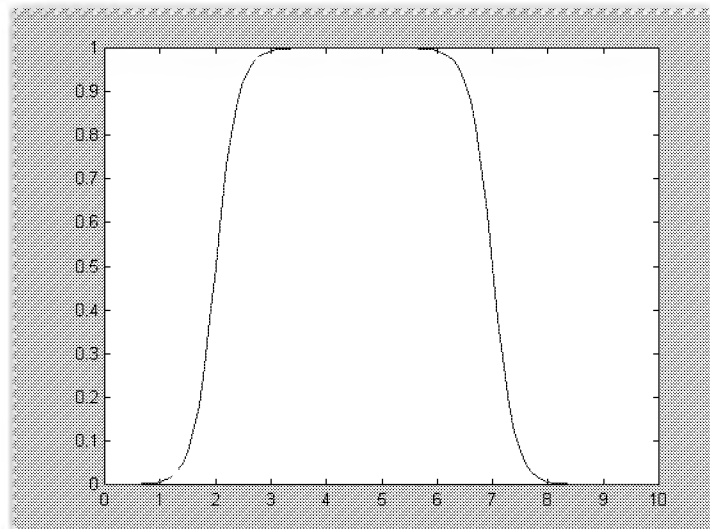


Рис. 5.22. Вид функции `dsigmf(x,[5 2 5 7])`

П р и м е р

```
» x = (0:0.1:10)';
» y1 = gauss2mf(x, [2 4 1 8]);
» y2 = gauss2mf(x, [2 5 1 7]);
» y3 = gauss2mf(x, [2 6 1 6]);
» y4 = gauss2mf(x, [2 7 1 5]);
» y5 = gauss2mf(x, [2 8 1 4]);
» plot(x, [y1 y2 y3 y4 y5]);
» set(gcf, 'name', 'gauss2mf', 'numbertitle', 'off');
```

3. Функция `gaussmf`.

Запись: `y = gaussmf(x,[sig c])`

О п и с а н и е. Задается функция принадлежности гауссова типа, зависящая от двух параметров (`[sig c]`).

П р и м е р

```
» x = 0:0.1:10;
» y = gaussmf(x,[2 5]);
» plot(x,y)
» xlabel('gaussmf, P=[2 5]')
```

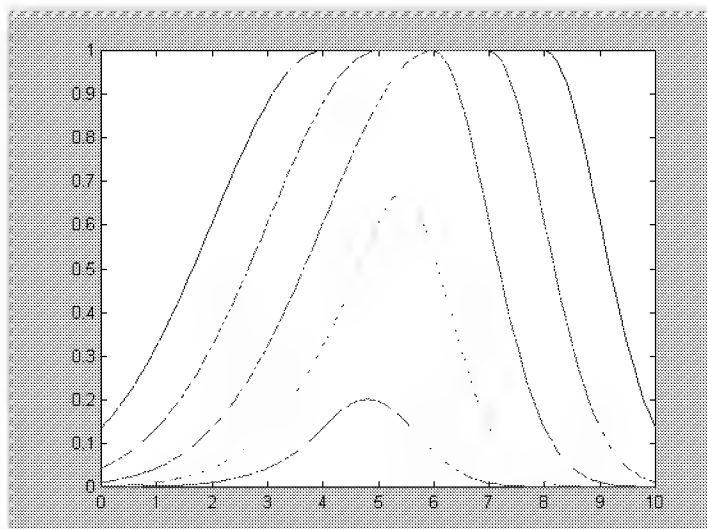


Рис. 5.25. Семейство кривых, определяемых функцией `gauss2mf`

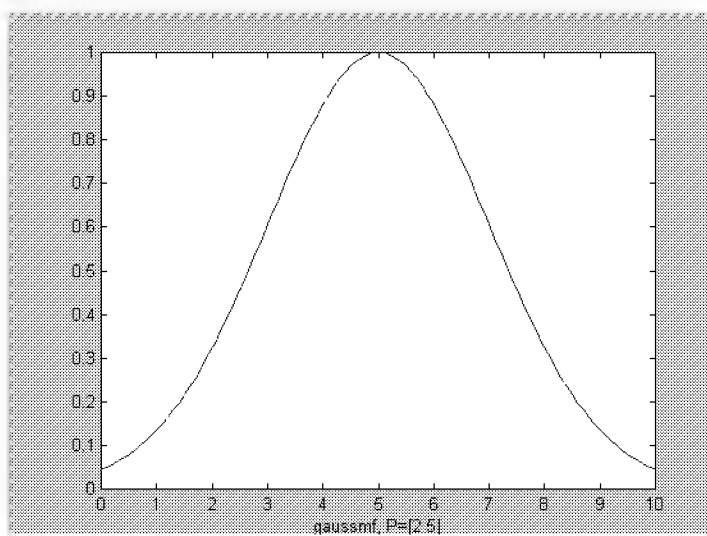


Рис. 5.24. Функция принадлежности типа гауссовой кривой

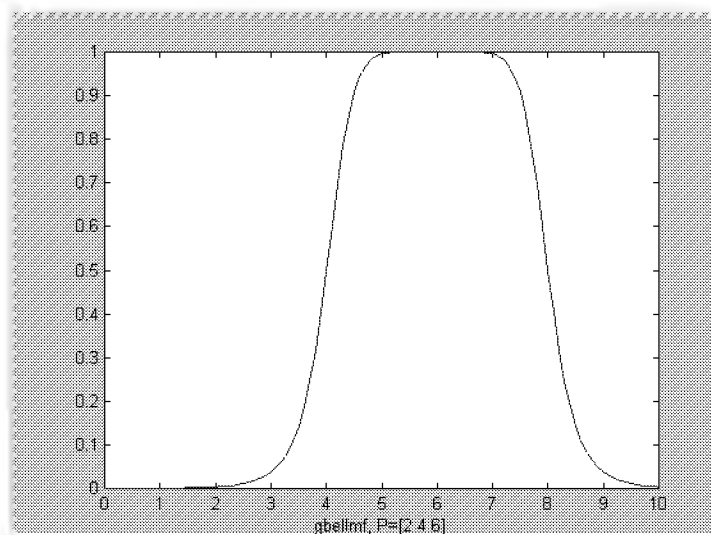
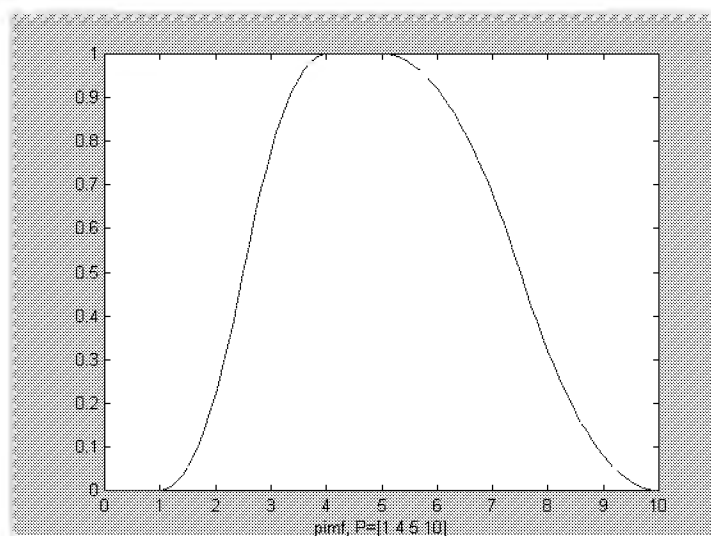


Рис. 5.26. График обобщенной колоколообразной функции

Рис. 5.26. График π -образной функции

4. Функция **gbellmf**.Запись: **y = gbellmf(x, params)**

Описание. Задается функция принадлежности так называемого обобщенного колоколообразного типа с аналитическим описанием, зависящим от трех числовых параметров:

$$(x, a, b, c) = \frac{1}{1 + |(x - c)/a|^{2b}}.$$

Пример

```
» x = 0:0.1:10;
» y = gbellmf(x,[2 4 6]);
» plot(x,y)
» xlabel('gbellmf, P = [2 4 6]')
```

5. Функция **pimf**.Запись: **y = pimf(x,[a b c d])**

Описание. Задается так называемую π -образная функция принадлежности, получившая свое название из-за своеобразной формы. Функция вычисляется с использованием сплайн аппроксимации по четырем точкам, задаваемым вектором параметров. Параметры a и d определяют основание кривой, а параметры b и c — положение плоской вершины.

Пример

```
» x = 0:0.1:10;
» y = pimf(x,[1 4 5 10]);
» plot(x,y)
» xlabel('pimf, P=[1 4 5 10]')
```

6. Функция **psigmf**.Запись: **y = psigmf(x,[a1 c1 a2 c2])**

Описание. Задается функция принадлежности, определяемая как произведение двух сигмоидальных функций

$$f_1(x, a_1, c_1) - f_2(x, a_2, c_2),$$

и зависящая от четырех параметров a_1, c_1, a_2, c_2 (вектора параметров $[a1\ c1\ a2\ c2]$).

Пример

```
» x = 0:0.1:10;
» y = psigmf(x,[2 3 -5 8]);
» plot(x,y)
» xlabel('psigmf, P = [2 3 -5 8]')
```

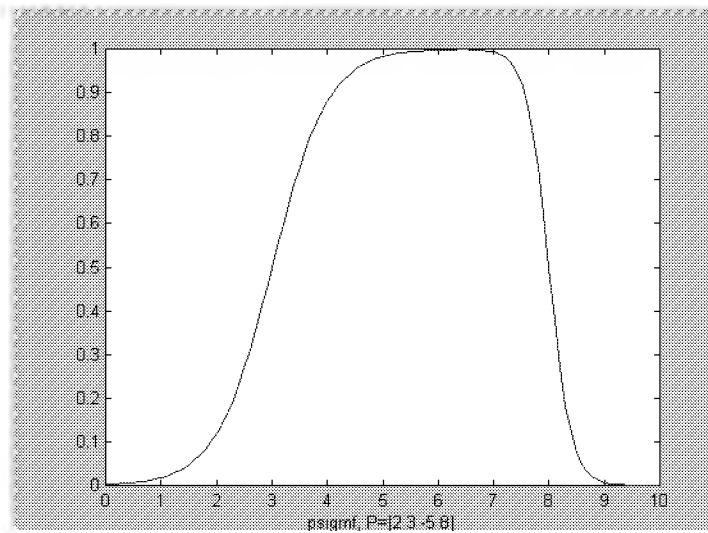
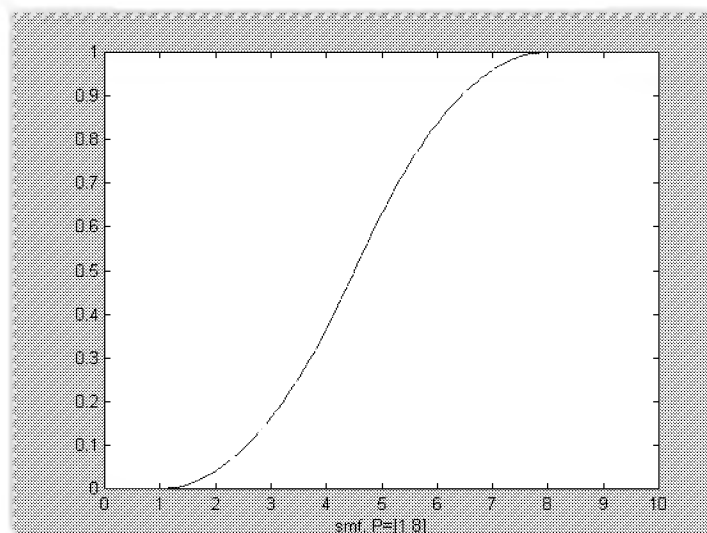
Рис. 5.27. Пример использования функции `psigmf`

Рис. 5.28. График S-образной функции

7. Функция **smf**.Запись: **y = smf(x,[a b])**

Описание. Задается зависящая от двух параметров так называемая S-образная функция принадлежности. Параметры a и b определяют диапазон значений аргумента, где функция возрастает.

Пример

```
» x = 0:0.1:10;
» y = smf(x,[1 8]);
» plot(x,y)
» xlabel('smf, P = [1 8]')
```

8. Функция **sigmf**.Запись: **y = sigmf(x,[a c])**

Описание. Задается так называемая сигмоидальная функция принадлежности, определяемая выражением, приведенным выше и зависящим от двух параметров.

Пример

```
» x = 0:0.1:10;
» y = sigmf(x,[2 4]);
» plot(x,y)
» xlabel('sigmf, P = [2 4]')
```

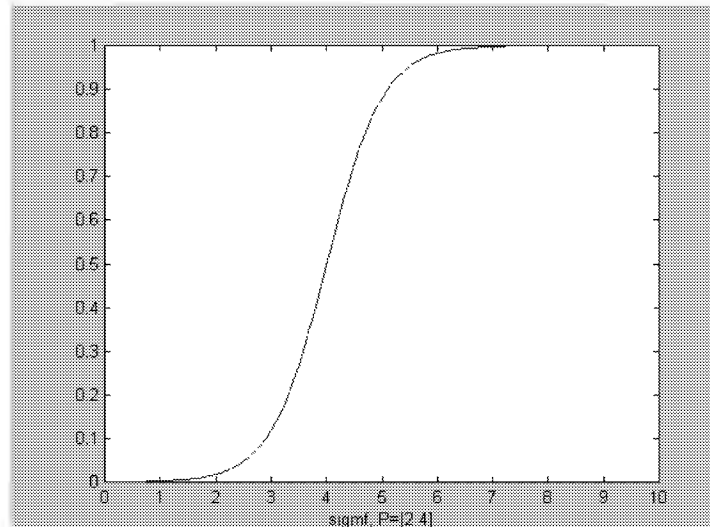


Рис. 5.29. График сигмоидальной функции

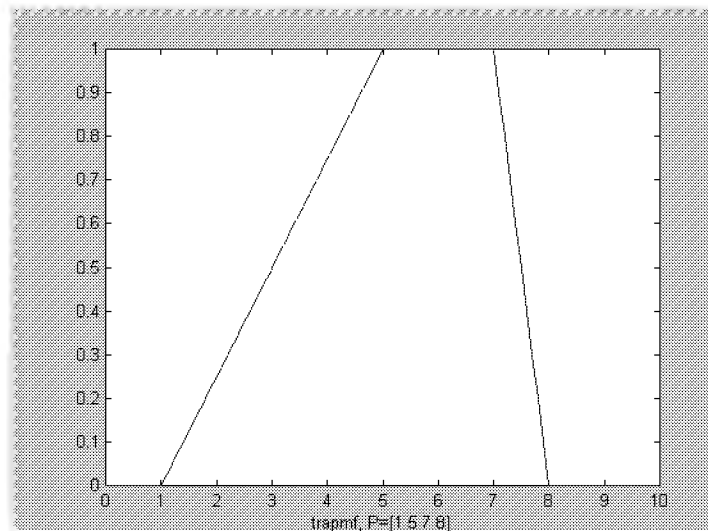
9. Функция **trapmf**.Запись: **y = trapmf(x,[a b c d])**

Рис. 5.30. Трапециевидальная функция принадлежности

Описание. Задается так называемая трапециевидальная функция принадлежности, зависящая от четырех параметров и определяемая формулой

$$f(x, a, b, c, d) = \left\{ \begin{array}{ll} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & b \leq x \leq c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & d \leq x \end{array} \right\},$$

при этом параметры a и d определяют основание кривой, a , b и c — положение вершины.

Пример

```
» x = 0:0.1:10;
» y = trapmf(x,[1 5 7 8]);
» plot(x,y)
» xlabel('trapmf, P = [1 5 7 8]')
```

10. Функция **trimf**.

Запись: **y = trimf(x,params)** или **y = trimf(x,[a b c])**

Описание. Задается зависящая от трех параметров функция принадлежности треугольной формы, при этом параметры *a* и *c* определяют основание треугольника, а параметр *b* — координату его вершины.

Пример

```
» x = 0:0.1:10;
» y = trimf(x,[3 6 8]);
» plot(x,y)
» xlabel('trimf, P=[3 6 8]')
```

11. Функция **zmf**.

Запись: **y = zmf(x,[a b])**

Описание. Задается зависящая от двух параметров функция принадлежности так называемой Z-образной формы. Параметры определяют диапазон убывания функции.

Пример

```
» x = 0:0.1:10;
» y = zmf(x,[3 7]);
» plot(x,y)
» xlabel('zmf, P = [3 7]')
```

5.5.4. Функции сохранения, открытия и использования созданной системы. Чтение, использование и сохранение на диске созданной системы нечеткого вывода в режиме командной строки осуществляется функциями

```
readfis('имя_файла'),  
evalfis(вектор_параметров, имя),  
writefis(имя) или writefis(имя, 'имя_файла')
```

Здесь **имя_файла** — наименование файла с записанной системой (без указания расширения), **имя** — идентификатор, который определен системе в рабочей среде MATLAB, **вектор_параметров** — набор значений входов, для которых требуется рассчитать выход (возможна и матрица параметров, тогда результат расчетов — вектор в случае одной выходной переменной или матрица при нескольких таких переменных).

Примеры (применительно к ранее созданной и сохраненной на диске в виде файла с именем **tip** экспертной системе для задачи о чаевых):

```
» readfis('tip');
» evalfis([1 2],a)
```

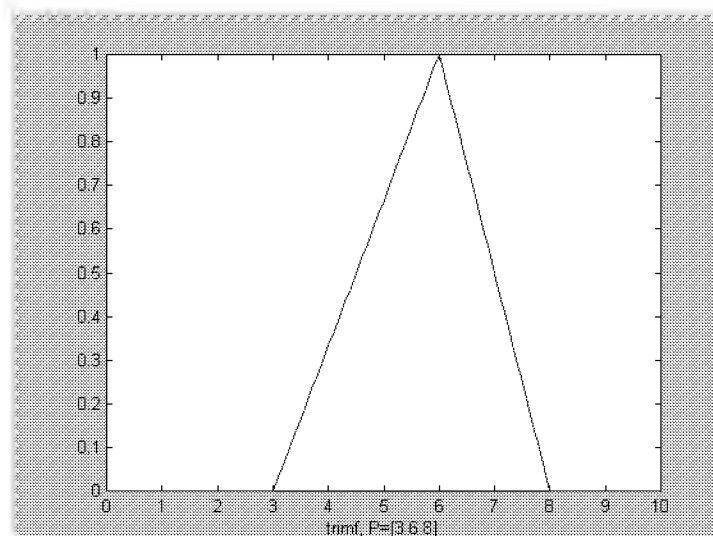


Рис. 5.31. График функции принадлежности треугольной формы

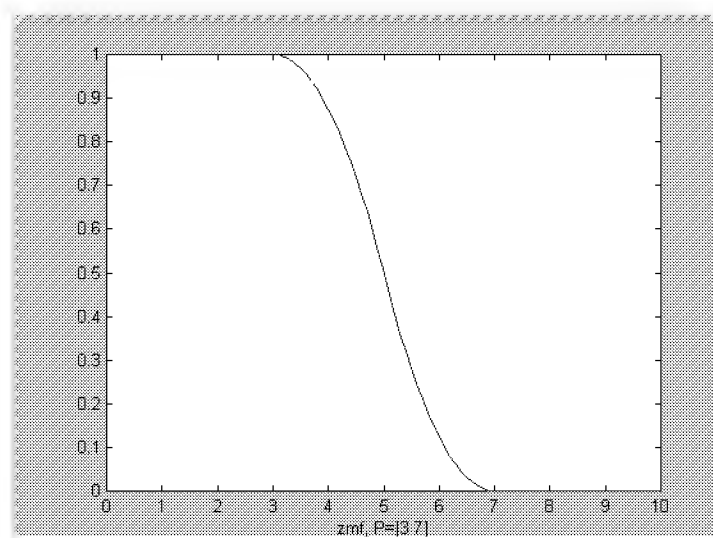


Рис. 5.32. Функция принадлежности Z-образной формы

```
ans =
    7.3949
» writefis(a)
ans =
tip
```

5.5.5. Функции использования графического окна. Следующие три функции позволяют использовать элементы графических изображений вне программ с графическим интерфейсом.

1. Команда (функция) **plotfis(a)** вызовет появление графического окна MATLAB с мнемоническим представлением разработанной системы (рис. 5.33).

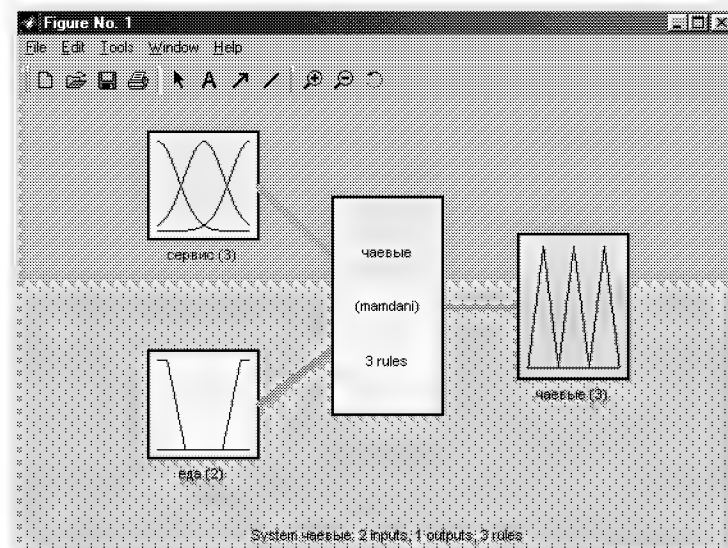
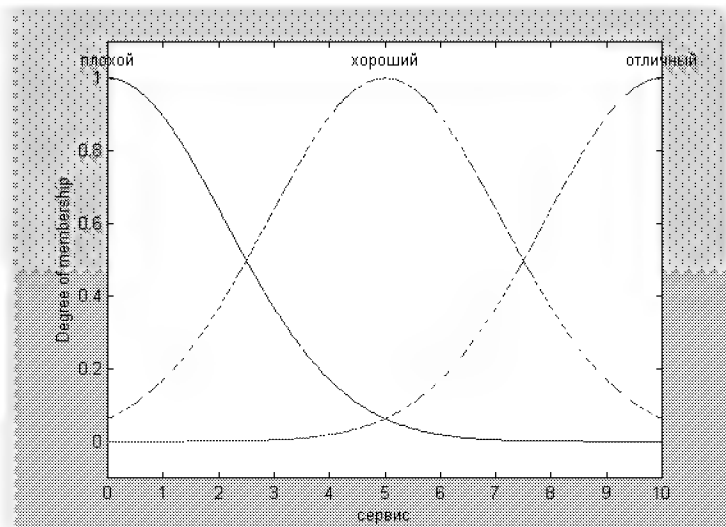
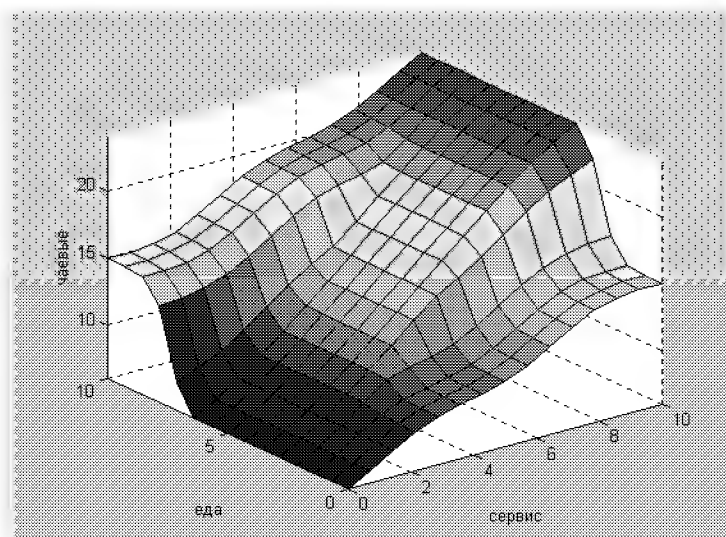


Рис. 5.33. Мнемоническое представление системы нечеткого вывода

2. Команда (функция) **plotmf** выполняет аналогичную операцию, но по отношению к графикам функций принадлежности. На рис. 5.34 приведен результат выполнения функции **plotmf(a,'input',1)**.

3. Наконец, команда (функция) **gensurf(a)** также делает то же самое, но применительно к поверхности отклика (рис. 5.35).

Рис. 5.34. Результат выполнения функции `plotmf(a,'input',1)`Рис. 5.35. Результат выполнения функция `gensurf(a)`

5.5.6. Функции создания, просмотра структуры и редактирования систем нечеткого вывода. Вообще, при желании можно совсем обойтись без программ графического интерфейса и, используя функции **newfis** (новая система), **addvar** (добавить переменную), **addmf** (добавить функцию принадлежности) и **addrule** (добавить правило), сконструировать систему нечеткого вывода целиком в режиме командной строки MATLAB. Процесс этот, надо сказать, значительно более трудоемкий, чем с применением указанных программ, поэтому, не вдаваясь особенно в детали, приведем лишь пример построения такой системы в задаче о чаевых:

```
a = newfis('tip');
a = addmf(a,'input',1,'сервис',[0 10]);
a = addmf(a,'input',1,'плохой','gaussmf',[1.5 0]);
a = addmf(a,'input',1,'хороший','gaussmf',[1.5 5]);
a = addmf(a,'input',1,'отличный','gaussmf',[1.5 10]);
a = addvar(a,'input','еда',[0 10]);
a = addmf(a,'input',2,'подгоревшая','trapmf',[-2 0 1 3]);
a = addmf(a,'input',2,'превосходная','trapmf',[7 9 10 12]);
a = addvar(a,'output','tip',[0 30]);
a = addmf(a,'output',1,'малые','trimf',[0 5 10]);
a = addmf(a,'output',1,'средние','trimf',[10 15 20]);
a = addmf(a,'output',1,'щедрые','trimf',[20 25 30]);
ruleList = [...
1 1 1 1 2
2 0 2 1 1
3 2 3 1 2];
a = addrule(a,ruleList);
```

При необходимости после просмотра элементов структуры (системы с идентификатором **a**) в режиме командной строки следует ввести команду вида **a.метка**, например,

```
» a.type
Получим ответ:
ans =
mamdani
```

Получение информации по всем элементам структуры обеспечивается функцией **getfis(a)**.

Результат ее выполнения:

```
Name = tip
Type = mamdani
NumInputs = 2
InLabels =
сервис
еда
NumOutputs = 1
OutLabels =
```

```

чаевые
NumRules = 3
AndMethod = min
OrMethod = max
ImpMethod = min
AggMethod = max
DefuzzMethod = centroid

```

```
ans =
```

```
tip
```

Функция **setfis** в определенном смысле противоположна функции **getfis** и позволяет изменять метки. Пример выполнения функции:

```

» a = setfis(a,'name','чаевые')
a =
name: 'чаевые'
type: 'mandani'
andMethod: 'min'
orMethod: 'max'
defuzzMethod: 'centroid'
impMethod: 'min'
aggMethod: 'max'
input: [1×2 struct]
output: [1×1 struct]
rule: [1×3 struct]

```

Полный просмотр структуры системы нечеткого вывода осуществляется функцией **showfis(a)**.

Просмотр правил, включенных в систему нечеткого вывода, осуществляется с помощью функции **showrule**.

Запись:

```

showrule(имя)
showrule(имя,номера_правил)
showrule(fis,номера_правил,формат)
showrule(fis,номера_правил,формат,язык)

```

Описание. В функции **имя** — это идентификатор рассматриваемой системы в среде MATLAB,

номера_правил — список правил, которые нужно показать, **формат** — строковая переменная, имеющая значения **'verbose'**, **'symbolic'** или **'indexed'**, **язык** — строковая переменная со значениями **'english'**, **'français'** или **'deutsch'** (установки по умолчанию — **'verbose'** и **'english'**).

Примеры

```

» a = readfis('tip');
» showrule(a,1)
ans =

```

```

1. If (сервис is плохой) or (еда is подгоревшая) then
(чаевые is малые) (1)
    » showrule(a,[3 1],'symbolic')
    ans =
3. (сервис==отличный)|(еда==превосходная)⇒
(чаевые=щедрые) (1)
1. (сервис==плохой)|(еда==подгоревшая)⇒
(чаевые=малые) (1)

```

Функция **rmmf** используется для удаления функции принадлежности из состава системы.

Запись:

```
имя = rmmf(имя,'varType',varIndex,'mf',mfIndex)
```

О п и с а н и е. Параметры функции: **имя** — идентификатор системы в среде MATLAB, **varType** — строковая переменная со значениями **'input'** или **'output'**, **varIndex** — порядковый номер переменной (по списку переменных входа или выхода), **mf** — строковая переменная, задающая функцию принадлежности, **mfIndex** — порядковый данной функции.

Функция **rmvar** удаляет переменную из состава системы.

Запись:

```
[новое_имя,errorStr] = rmvar(имя,'varType',varIndex)
новое_имя = rmvar(имя,'varType',varIndex)
```

О п и с а н и е. Здесь переменные **varType** и **varIndex** имеют тот же смысл, что и в предыдущей функции, переменная **errorStr** позволяет записать любое сообщение об ошибке, **новое_имя** — новое имя системы с исключенной переменной.

Функция приведения к четкости (дефазификации) **defuzz** позволяет по заданной функции принадлежности определить соответствующее четкое значение.

Запись: **out = defuzz(x,mf,type)**

О п и с а н и е. Здесь **x** — обозначение числового аргумента, **mf** — тип функции принадлежности, **type** — задаваемый метод приведения к четкости — строковая переменная, имеющая возможные значения:

- **centroid;**
- **bisector;**
- **mom;**
- **som;**
- **lom.**

Смысл этих значений был рассмотрен ранее.

П р и м е р

```
» x = -10:0.1:10;
```



```
» mf = trapmf(x,[-10 -8 -4 7]);
» xx = defuzz(x,mf,'centroid')
xx =
    -3.2857
```

Функция **evalmf** вычисляет значения заданной функции принадлежности.

Запись: **y = evalmf(x,mfParams,mfType)**

Описание. Здесь **x** — обозначение числового аргумента, **mfParams** — вектор необходимых числовых параметров, **mfType** — тип функции принадлежности.

Пример (см. рис. 5.36)

```
» x = 0:0.1:10;
» mfparams = [2 4 6];
» mftype = 'gbellmf';
» y = evalmf(x,mfparams,mftype);
» plot(x,y)
» xlabel('gbellmf, P = [2 4 6]')
```

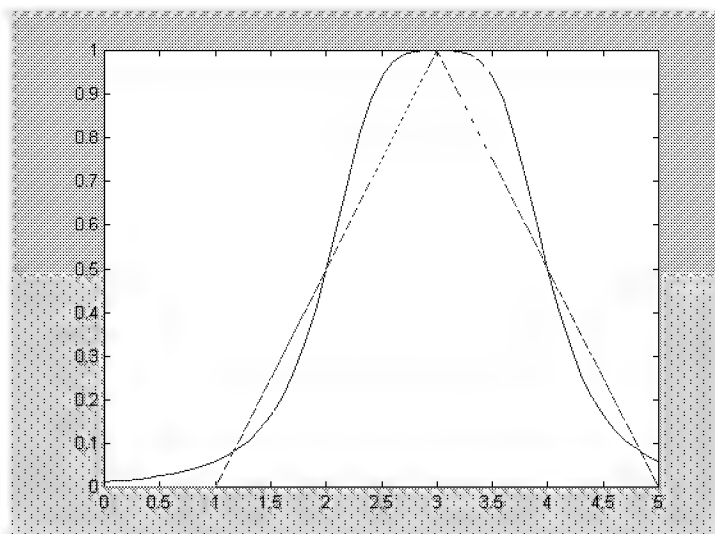


Рис. 5.36. Графическое представление результата выполнения команды **y = evalmf(x,mfParams,mfType)**

Функция **mf2mf** позволяет заменять одну функцию принадлежности близкой ей по форме другой с некоторым «эквивалентным» набором числовых параметров.

Запись: **outParams = mf2mf(inParams,inType,outType)**

Описание. **inParams** — набор параметров исходной функции, **inType** — тип исходной функции принадлежности, **outType** — тип эквивалентной функции принадлежности, **outParams** — набор преобразованных параметров.

Пример

```
» x = 0:0.1:5;
» mfp1 = [1 2 3];
» mfp2 = mf2mf(mfp1,'gbellmf','trimf')
```

mfp2 =

```
1 3 5
» plot(x,gbellmf(x,mfp1),x,trimf(x,mfp2))
```

Функцию **parsrule** можно отнести к числу сервисных.

Запись:

новое_имя = parsrule(имя,текст_правил)

новое_имя = parsrule(имя, текст_правил,формат)

новое_имя = parsrule(имя, текст_правил,формат,язык)

Описание. В данной функции идентификаторы **имя** и **новое_имя** относятся к исходной и модернизированной системам

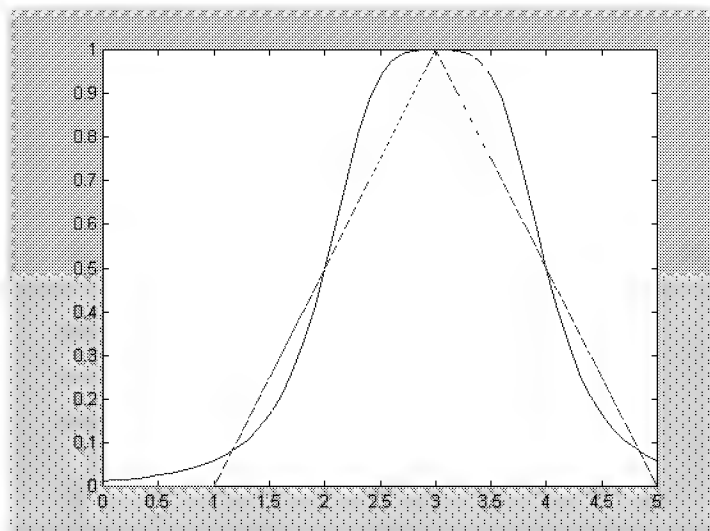


Рис. 5.37. Графики исходной (колоколообразной) и эквивалентной (треугольной) функций принадлежности

с нечетким выводом, строковые переменные **формат** и **язык** имеют тот же смысл, что и в рассмотренной ранее функции **showrule**,

текст правил заданный в текстовой форме набор правил системы (на языке, определенной переменной **язык**; по умолчанию английский, что допускает использование, в частности, русских имен для переменных, но требует английских связующих слов if, and, or, then, is). Функция выполняет грамматический разбор исходного текста и выдает набор нечетких правил в указанном формате (по умолчанию — подробный).

5.5.7. Функция создания и/или обучения гибридных сетей с архитектурой ANFIS. Функция **anfis** записывается в виде:

```
[имя,ошибка_о,шаг] = anfis(trnData)
[имя,ошибка_о,шаг] = anfis(trnData,имя)
[имя1,ошибка_о,шаг] = ...
anfis(trnData,имя,trnOpt,dispOpt)
[имя1,ошибка_о,шаг,имя2,ошибка_п] = ...
anfis(trnData,trnOpt,dispOpt,chkData)
[имя1,ошибка_о,шаг,имя2,ошибка_п] = ...
anfis(trnData,trnOpt,dispOpt,chkData,optMethod)
```

Функция предназначена для создания и/или обучения гибридных сетей с архитектурой ANFIS. Значения аргументов функции:

- **trnData** матрица данных для обучения сети (обучающая выборка); последний столбец соответствует (единственной) выходной переменной, остальные столбцы — входным переменным;
- **имя** идентификатор создаваемой гибридной сети; если структура системы нечеткого вывода с таким идентификатором уже создана, то она будет использована для настройки числовых параметров, в противном случае структура будет создана при выполнении функции с опциями, по умолчанию соответствующими выполнению функции **genfis1** (см. ниже);
- **trnOpt** вектор опций обучения с элементами:
 - trnOpt(1):** количество циклов обучения (по умолчанию 10),
 - trnOpt(2):** целевой уровень ошибки обучения (по умолчанию 0),
 - trnOpt(3):** начальный шаг алгоритма обучения (по умолчанию 0.01),
 - trnOpt(4):** коэффициент уменьшения шага (по умолчанию 0.9),
 - trnOpt(5):** коэффициент увеличения шага (по умолчанию 1.1);
- **dispOpt** вектор опций вида выводимой информации (по умолчанию все элементы единичные, что означает вывод всех видов возможной информации в процессе выполнения функции) с элементами:

dispOpt(1): ANFIS-информация, такая, как число входов, функции принадлежности и т. п.,

dispOpt(2): ошибка,
dispOpt(3): шаг обновления (корректировки) по каждому параметру,

dispOpt(4): конечные результаты;

- **chkData** матрица проверочных данных (проверочная выборка), имеющая такой же набор столбцов, что и матрица данных для обучения сети, но, вообще говоря, другое число строк;

- **optMethod** параметр, определяющий вид алгоритма обучения гибридной системы (1 — гибридный алгоритм, 0 — алгоритм обратного распространения ошибки; по умолчанию — 1).

Процесс обучения сети заканчивается, когда выполнено заданное число циклов обучения или ошибка обучения уменьшилась до заданного уровня. При отсутствии некоторых аргументов их значения принимаются по умолчанию.

Выходные параметры функции:

- **ошибка_о** — массив (вектор) значений ошибок для обучающей выборки;

- **ошибка_п** — массив (вектор) значений ошибок для проверочной выборки;

- **шаг** — массив значений шага в алгоритме обучения;

- **имя1** — идентификатор (имя) сети, образованной исходя из минимальной ошибки обучения;

- **имя2** — идентификатор (имя) сети, образованной исходя из минимальной ошибки для проверочной выборки.

Пример 1

```
» x = (0:0.1:10)';
```

```
» y = sin(2*x)./exp(x/5);
```

```
» trnData = [x y];
```

```
» a = anfis(trnData);
```

ANFIS info:

Number of nodes: 12

Number of linear parameters: 4

Number of nonlinear parameters: 6

Total number of parameters: 10

Number of training data pairs: 101

Number of checking data pairs: 0

Number of fuzzy rules: 2

Start training ANFIS ...

```
1 0.313942
```

```
2 0.31388
```

```
3 0.313817
```

```
4 0.313753
```

```
5 0.313689
```

Step size increases to 0.011000 after epoch 5.

```

6    0.313624
7    0.313552
8    0.313479
9    0.313406
Step size increases to 0.012100 after epoch 9.
10   0.313332
Designated epoch number reached → ANFIS training
completed at epoch 10.
» plot(x,y,x,evalfis(x,a),'-');
» legend('Обучающая выборка','Выход ANFIS');

Пример 2
» x = (0:0.1:10)';
» y = sin(2*x)./exp(x/5);
» trnData = [x y];
» numMFs = 5;
» mfType = 'gbellmf';
» epoch_n = 20;
» in_fismat = genfis1(trnData,numMFs,mfType);
» out_fismat = anfis(trnData,in_fismat,20);
ANFIS info:
Number of nodes: 24
Number of linear parameters: 10
Number of nonlinear parameters: 15
Total number of parameters: 25
Number of training data pairs: 101
Number of checking data pairs: 0
Number of fuzzy rules: 5
Start training ANFIS ...
1    0.0694086
2    0.0680259
3    0.066663
4    0.0653198
5    0.0639961
Step size increases to 0.011000 after epoch 5.
6    0.0626917
7    0.0612787
8    0.0598881
9    0.0585193
Step size increases to 0.012100 after epoch 9.
10   0.0571712
11   0.0557113
12   0.0542741
13   0.052858
Step size increases to 0.013310 after epoch 13.
14   0.0514612
15   0.0499449
16   0.0484475
17   0.0469667

```

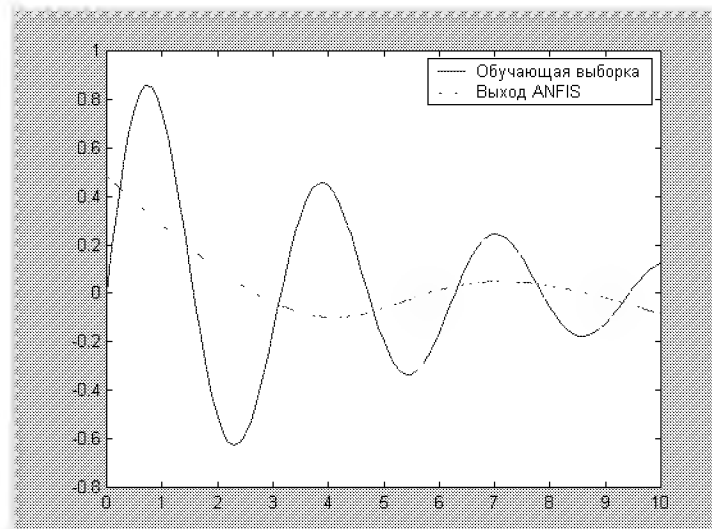


Рис. 5.38. Обучающая выборка и выход системы **a**

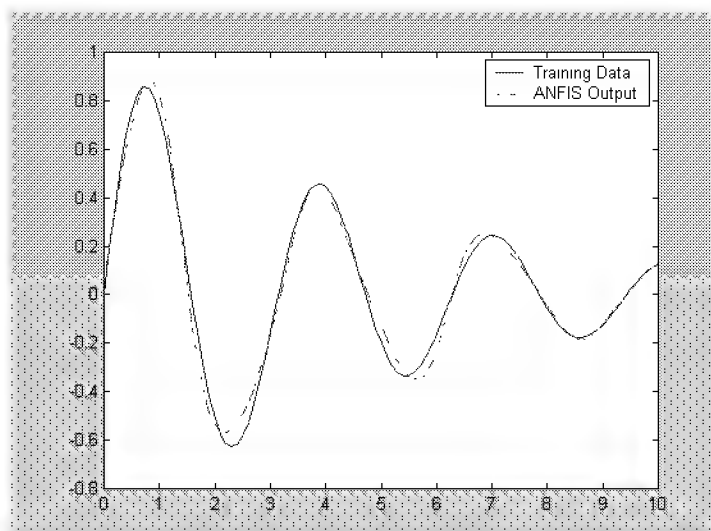


Рис. 5.39. Обучающая выборка и выход системы **fismat**

Step size increases to 0.014641 after epoch 17.

18 0.0455003

19 0.0439022

20 0.0423184

Designated epoch number reached → ANFIS training completed at epoch 20.

```
» plot(x,y,x,evalfis(x,out_fismat),'-');
```

```
» legend('Training Data','ANFIS Output');
```

В примере 2 при проектировании гибридной системы заранее были заданы некоторые значения (количество и тип функций принадлежности, количество циклов обучения), в примере 1 для той же обучающей выборки все установки при проектировании системы выбраны по умолчанию. Отличие в качестве функционирования двух систем наглядно иллюстрируются рис. 5.38 и 5.39.

5.5.8. Функция кластеризации. Функция **fcm** осуществляет кластеризацию с использованием алгоритма Fuzzy c-means (см. выше). Она записывается в виде:

```
[center,U,obj_fcn] = fcm(data,cluster_n)
```

```
[center,U,obj_fcn] = fcm(data,cluster_n,options)
```

О п и с а н и е. Аргументы функции:

- **data** — матрица данных, каждый столбец которой соответствует одной из переменных, а каждая строка — одному из опытов;
- **cluster_n** — число задаваемых кластеров (должно быть больше 1);
- **options** — вектор опций функции с элементами:
 - options(1):** показатель для матрицы **U** (по умолчанию 2.0),
 - options(2):** максимальное число итераций при определении центров кластеров (по умолчанию 100),
 - options(3):** минимальная сумма улучшения (по умолчанию 1e-5),
 - options(4):** вывод информации в процессе итераций (по умолчанию 1).

Выходные параметры функции:

- **center** — матрица, элементы которой (по столбцам) являются координатами найденных центров кластеров, число строк равно числу центров;
- **U** — матрица значений принадлежности данных к выявленным кластерам;
- **obj_fcn** — значения целевой функции в процессе итераций.

Пример

```
» load clusterdemo.dat;
» data = clusterdemo;
» [center,U,obj_fcn] = fcm(data, 3);
Iteration count = 1, obj. fcn = 55.941585
Iteration count = 2, obj. fcn = 42.971277
Iteration count = 3, obj. fcn = 42.764109
Iteration count = 4, obj. fcn = 41.688335
Iteration count = 5, obj. fcn = 37.070571
Iteration count = 6, obj. fcn = 29.262573
Iteration count = 7, obj. fcn = 22.996848
Iteration count = 8, obj. fcn = 16.162985
Iteration count = 9, obj. fcn = 15.443897
Iteration count = 10, obj. fcn = 15.430759
Iteration count = 11, obj. fcn = 15.430533
Iteration count = 12, obj. fcn = 15.430528
» center
center =
    0.2051    0.5960    0.8090
    0.7939    0.7892    0.1968
    0.5963    0.1923    0.5057
```

5.5.9. Функция генерации FIS-структуры. Функция **genfis1** генерирует FIS-структуру по экспериментальным данным без проведения их кластеризации:

```
имя = genfis1(data)
имя = genfis1(data,numMFs,inmftype, outmftype)
```

Описание. Функцией генерируется структура системы нечеткого вывода типа Sugeno, являющаяся исходной для последующего формирования и обучения гибридной системы с помощью функции **anfis**. Аргументы функции:

- **data** — матрица данных (обучающая выборка), последний столбец которой соответствует выходной переменной, а остальные — входным переменным и число строк в которой равно числу наборов экспериментальных данных (образцов);
- **numMFs** — вектор, элементы которого определяют число функций принадлежности, задаваемых для каждого входа; если для всех входов задается одно и то же число таких функций, данный аргумент задается как скаляр (один элемент);
- **inmftype** — строковый массив, элементы которого типы функций принадлежности, задаваемые для входных переменных;
- **outmftype** — строковая переменная, определяющая тип функций принадлежности выходной переменной (возможны только значения **'linear'** или **'constant'**).

Число функций принадлежности выходной переменной равно числу нечетких правил, генерируемых **genfis1**, и зависит от элементов вектора **numMFs**. По умолчанию **numMFs = 2**, **inmfType = 'gbellmf'**. Значения по умолчанию устанавливаются, если функция применяется без трех последних аргументов.

Применение функции ограничено размерностью задачи: так, при N входных переменных с минимальным разбиением области определения каждой переменной на две подобласти будет генерироваться 2^N правил, что при числе входов больше 5-6 делает анализ этих правил практически невозможным. Более экономной в этом плане является функция **genfis2** (см. ниже).

Пример

```
» data = [rand(10,1) 10*rand(10,1)-5 rand(10,1)];
» numMFs = [3 7];
» mfType = str2mat('pimf','trimf');
» fismat = genfis1(data,numMFs,mfType);
» [x,mf] = plotmf(fismat,'input',1);
» subplot(2,1,1), plot(x,mf);
» xlabel('input 1 (pimf)');
» [x,mf] = plotmf(fismat,'input',2);
» subplot(2,1,2), plot(x,mf);
» xlabel('input 2 (trimf)');
```

См. также пример 2 для функции **anfis**.

5.5.10. Функция генерации структуры нечеткого вывода. Функция **genfis2** генерирует структуру системы нечеткого вывода (типа Sugeno) с использованием алгоритма Subtractive clustering кластеризации данных:

```
имя = genfis2(Xin,Xout,radii)
имя = genfis2(Xin,Xout,radii,xBounds)
имя = genfis2(Xin,Xout,radii,xBounds,options)
```

Аргументами рассматриваемой функции являются:

- **Xin** — матрица (массив) входных данных обучающей выборки, столбцы которой ассоциированы с входными переменными, а каждая строка — с отдельным опытом;

- **Xout** — матрица выходных переменных обучающей выборки, столбцы которой представляют значения данных переменных, а число строк равно числу строк матрицы **Xin**;

- **radii** (радиусы) — вектор, определяющий «области влияния» центров кластеров по каждой входной переменной (в предположении, что область определения данных переменных — многомерный параллелепипед) с неотрицательными элементами, меньшими единицы; пусть, например, число входов — 3, тогда **radii =**

$= [0.5 \ 0.4 \ 0.3]$ означает, что по первой переменной «область влияния» составляет 0.5 от диапазона изменения этой переменной, а по второй и третьей соответственно 0.4 и 0.3 (эти диапазоны определяются по элементам столбцов матрицы **Xin**). Если рассматриваемый аргумент задан как скаляр, то по всем переменным устанавливается один и тот же размер «области влияния» кластеров. Параметр имеет важное значение для проведения разбиения области определения входов на подобласти с последующим установлением числа нечетких правил;

- **XBounds** — матрица с 2 строками и N столбцами (N — число столбцов в матрице **Xin**, т.е. число входных переменных), устанавливающая границы областей определения входных переменных. Данные этой матрицы используются для преобразования (масштабирования) входов к диапазону $[-1, 1]$. Например, **Xbounds** = $[-10 \ 0 \ -1; 10 \ 50 \ 1]$ задает диапазон $[-10, 10]$ для первой переменной, $[0, 50]$ для второй и $[-1, 1]$ для третьей. Если рассматриваемый аргумент опущен, границы областей определяются как максимальные и минимальные элементы по столбцам матрицы **Xin**;

- **options** — вектор опций, устанавливающий параметры алгоритма кластеризации (подробней см. при описании функции **subclust**).

Функция **genfis2**, так же как **genfis1**, может применяться в комплексе с функцией **anfis** (для настройки параметров системы нечеткого вывода), но, в отличие от последней, **genfis2** возвращает уже полностью определенную, готовую к использованию систему, не требующую, вообще говоря, дополнительной оптимизации.

Функция **genfis2** производит более экономное, чем **genfis1** разбиение пространства входов на подобласти и может, поэтому, являться эффективным инструментом для выявления правил из набора экспериментальных данных.

Примеры

```
a = genfis2(Xin,Xout,0.5)
```

```
a = genfis2(Xin,Xout,[0.5 0.25 0.3])
```

5.5.11. Функция возврата центров кластеров. Функция **subclust** возвращает центры кластеров.

Запись: **[C,S] = subclust(X,radii,xBounds,options)**

Функцией определяются центры кластеров набора экспериментальных данных, отражаемых матрицей **X** (столбцы матрицы соответствуют переменным, строки — экспериментальным «точкам» или опытам) на основе реализации «вычитающего» алгоритма кластеризации (Subtractive clustering). В его основе лежит пред-

положение, что каждая экспериментальная точка может быть центром кластера, при этом вначале для каждой точки вычисляется

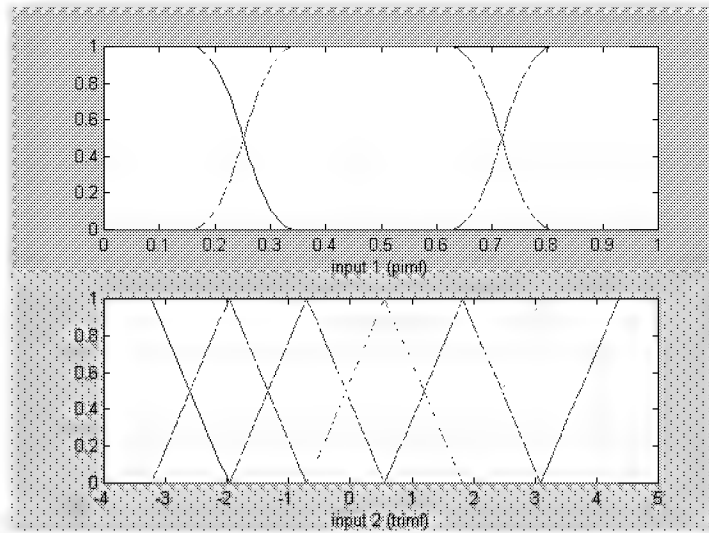


Рис. 5.40. Функции принадлежности, определенные `genfis1`

мера правдоподобия данного предположения («потенциал точки»), основанная на плотности точек в заданной окрестности рассматриваемой. Дальнейшие вычисления происходят итеративно:

- точка с наибольшим потенциалом объявляется центром первого кластера;
- из отмеченной окрестности этой точки удаляются все остальные точки;
- из оставшихся точек объявляется центр следующего кластера и т. д., пока не будут рассмотрены (исключены или объявлены центрами кластеров) все точки.

Размеры окрестностей задаются элементами вектора **radii**, который (как и аргумент **xBounds**) аналогичен рассмотренному при описании функции **Genfis2**.

Аргумент **options** является вектором со следующими элементами:

- **options(1) = quashFactor** (фактор подавления): используется для расширения «сферы влияния» выделенного центра кластера для «подавления» (уменьшения) потенциала точек, относящихся к данному кластеру; по умолчанию равен 1.25;

- **options(2) = acceptRatio** (коэффициент принятия): устанавливает, больше какого значения по отношению к потенциалу принятого центра кластера должен быть потенциал точки, чтобы эту точку можно было объявить центром нового кластера; по умолчанию 0.5;

- **options(3) = rejectRatio** (коэффициент отклонения): устанавливает, меньше какого значения по отношению к потенциалу принятого центра кластера должен быть потенциал точки, чтобы эту точку нельзя было объявить центром нового кластера; по умолчанию 0.15;

- **options(4) = verbose** (подробности): если этот элемент ненулевой, то выдается подробная информация и процессе нахождения центров кластеров; по умолчанию 0.

Функция возвращает центры кластеров в виде матрицы **C**; каждая строка этой матрицы содержит координаты одного из найденных центров. Вектор **S** содержит элементы, определяющие диапазоны влияния центров кластеров по каждой переменной (для всех центров эти диапазоны одинаковы).

Пример

```
» X = load('clustdemo.dat');
```

```
» [C,S] = subclust(X,0.5)
```

C =

```
0.2220  0.5937  0.8113
```

```
0.7797  0.8191  0.1801
```

```
0.5914  0.1721  0.4872
```

S =

```
0.1904  0.1988  0.2251
```

5.5.12. Различные другие функции. К этой группе относятся 10 функций:

1) **convertfis** — обеспечивает конвертацию FIS-матрицы пакета Fuzzy Logic версии 1.0 в FIS-структуру пакета Fuzzy Logic версии 2.0:

```
имя_новое = convertfis(имя_старое)
```

2) **discfis** — преобразует непрерывные функции принадлежности системы нечеткого вывода в дискретные (задаваемые наборами точек).

3) **evalmmf** — вычисляет значения одновременной нескольких функций принадлежности. Является многомерным аналогом функции **evalmf**.

4) **fstrvcats** — объединяет заданные векторы и матрицы в одну матрицу:

```
Y = fstrvcats(y1,y2,y3,...)
```

Функция возвращает матрицу **Y**, представляющую собой объединение матриц-аргументов **y1,y2,y3,...**, при этом для выравнивания размеров итоговой матрицы она дополняется нулями. Элементы аргументов могут быть строкового типа, при этом их символы преобразуются в числовую форму.

5) **fuzarith** — выполняет алгебраические операции над нечеткими множествами:

C = fuzarith(X, A, B, operator)

Здесь **X** — вектор, выполняющий роль универсального множества, **A** и **B** — векторы, определяющие нечеткие подмножества-операнды, **C** — вектор, определяющий нечеткое подмножество

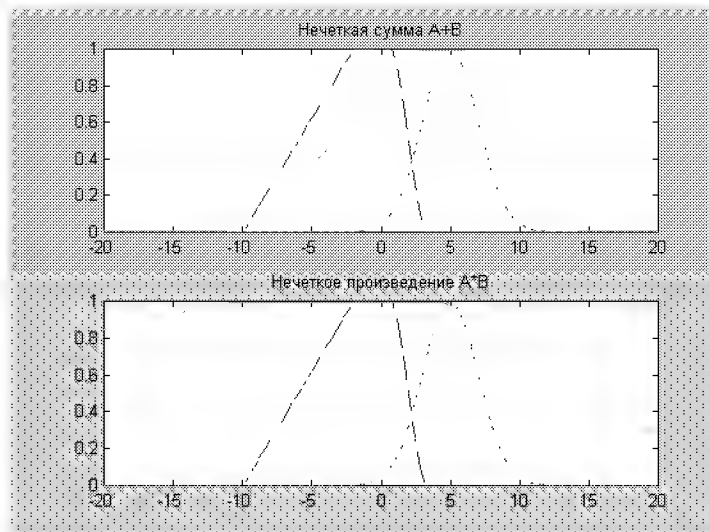


Рис. 5.41. Результаты выполнения функции **fuzarith**

результат операции, **operator** — строковая переменная, задающая вид операции и имеющая одно из возможных значений **'sum'** (сумма), **'sub'** (вычитание), **'prod'** (умножение), **'div'** (деление).

Пример

```
» point_n = 101; % Задание количества точек универсального
множества
» x = linspace(min_x, max_x, point_n)';
» A = trapmf(x, [-10 -2 1 3]); % Трапецидальная функция
принадлежности подмножества A
» B = gaussmf(x, [2 5]); % Гауссова функция принадлежности
подмножества B
```

```

» C1 = fuzarith(x, A, B, 'sum');
» subplot(2,1,1);
» plot(x, A, 'b--', x, B, 'm:', x, C1, 'c');
» title('Нечеткая сумма A+B');
» C2 = fuzarith(x, A, B, 'prod');
» subplot(2,1,2);
» plot(x, A, 'b--', x, B, 'm:', x, C2, 'c');
» title('Нечеткое произведение A*B');

```

6) **findrow** — возвращает номера аргументов функции **fstrvcat** по их именам.

7) **genparam** — возвращает параметры заданных функций принадлежности, определяя их по набору входных экспериментальных данных. Данные результаты могут быть использованы как начальные приближения при последующем применении команды **anfis**.

8) **mam2sug** — преобразует систему нечеткого вывода с алгоритмом Mamdani в систему, использующую алгоритм Sugeno:

```
имя_sug = mam2sug(имя_mam)
```

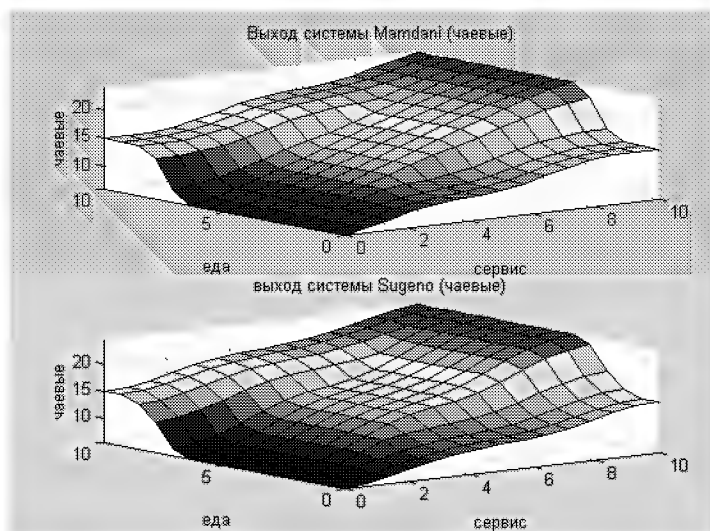


Рис. 5.42. Иллюстрация к выполнению функции **mam2sug**

Функция возвращает систему нечеткого вывода (с именем **имя_sug**), использующую алгоритм вывода Sugeno, по заданной

системе **имя_mam**, использующей алгоритм Mamdani. Возвращаемая система имеет постоянные функции принадлежности выходных переменных (допускается их число больше 1), определяемые как центроиды функций принадлежности следствий нечетких правил алгоритма Mamdani. Предпосылки правил при преобразовании не изменяются.

Пример

```
» a = readfis('tip');
» b = mam2sug(a);
» subplot(2,1,1); gensurf(a);
» title('Выход системы Mamdani (чаевые)');
» subplot(2,1,2); gensurf(b);
» title('выход системы Sugeno (чаевые)');
```

Замечание. Проверка, проведенная авторами, показала, что рассмотренная функция выполняется некорректно и приводит к неработающей системе вывода (попытки использования системы Sugeno вызывают сообщение об ошибке). Данный недостаток можно устранить, изменив в предпоследней строке файла `mam2sug.m` (в директории `Matlab/toolbox/fuzzy/fuzzy/`) `defuzzmethod` на `defuzzMethod`.

9) **probor** — выполняет операцию вероятностного ИЛИ над столбцами матрицы-аргумента:

Y = probor(X)

Если матрица-аргумент **X** имеет два столбца, **A** и **B**, то функция возвращает матрицу $Y = A + B - AB$; если аргумент содержит только один столбец, то $Y = X$.

10) **sugmax** позволяет определить максимально возможные диапазоны изменения выходных переменных в заданной системе нечеткого вывода, использующей алгоритм Sugeno.

5.5.13. Функции вызова диалоговых окон интерфейса. Данную группу образуют 12 функций:

1) **cmfdlg** — вызывает диалоговое окно для задания функции принадлежности пользователя,

2) **cmthdlg** — вызывает диалоговое окно для задания пользовательского алгоритма нечеткого вывода,

3) **fisgui** — позволяет генерировать элементы графического интерфейса пользователя,

4) **gfmfdlg** — вызывает диалоговое окно для задания числа и типа функций принадлежности при проектировании системы типа Sugeno (с последующим применением функции **anfis**),

- 5) **mfdlg** — вызывает диалоговое окно для задания дополнительной функции принадлежности,
- 6) **mfdrag** — обеспечивает «перетаскивание» функции принадлежности с использованием мыши,
- 7) **popundo** — возвращает из стека результаты отмены предыдущего действия,
- 8) **pushundo** — помещает в стек данные по отменяемому действию,
- 9) **savedlg** — вызывает диалоговое окно, выдающее запрос на сохранение результатов,

Рис. 5.43. Результат выполнения команды **gfmfdlg**

- 10) **statmsg** — помещает сообщения в поле состояния окна,
- 11) **updtfis** — обновляет элементы графического интерфейса пользователя,
- 12) **wsdlg** — вызывает диалоговое окно для выполнения операции сохранения в рабочем пространстве.

В качестве примера приведем выполнение функции **gfmfdlg**:

```
» a = readfis('tip');
» gfmfdlg('#init',a)
ans =
    '3 2'    'gaussmf'    'linear'
```


5.6. Работа Fuzzy Logic с блоками Simulink

Системы нечеткого вывода, созданные тем или иным образом с помощью пакета Fuzzy Logic Toolbox, допускают интеграцию с инструментами пакета Simulink, что позволяет выполнять моделирование систем в рамках последнего.

5.6.1. Пример: контроль уровня воды в баке. На рис. 5.44 изображен объект управления в виде бака с водой, к которому подходят две трубы: через одну трубу, снабженную краном, вода втекает в бак, через другую — вытекает.

Подачу воды в бак можно регулировать, больше или меньше открывая кран. Расход воды является неконтролируемым и зависит от диаметра выходной трубы (он фиксирован) и от текущего

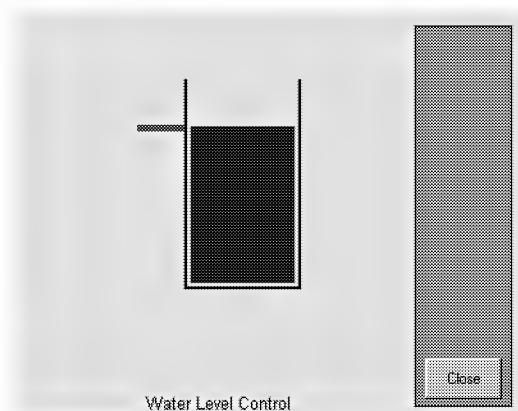


Рис. 5.44. Схематическое представление объекта управления (бака с водой)

уровня воды в баке. Если понимать под выходной (регулируемой) переменной уровень воды, а под регулирующим элементом — кран, то можно отметить, что подобный объект регулирования, с точки зрения его математического описания, является динамическим и существенно нелинейным.

Определим цель управления здесь как установление уровня воды в баке на требуемом (изменяющемся) уровне и попробуем решить соответствующую задачу управления средствами нечеткой логики.

Очевидно, в регулятор, обеспечивающий достижение цели управления, должна поступать информация о несоответствии (разности) требуемого и фактического уровней воды, при этом данный

регулятор должен вырабатывать управляющий сигнал на регулируемый элемент (кран).

В первом приближении функционирование регулятора можно описать набором из следующих правил:

1. If (level is okay) then (valve is no_change) (1)
2. If (level is low) then (valve is open_fast) (1)
3. If (level is high) then (valve is close_fast) (1)
4. If (level is okay) and (rate is positive) then (valve is close_slow) (1)
5. If (level is okay and (rate is negative) then (valve is open_slow) (1),

что в переводе означает:

1. Если (уровень соответствует заданному), то (кран без изменения) (1)
2. Если (уровень низкий), то (кран быстро_открыть) (1)
3. Если (уровень высокий), то (кран быстро_заккрыть) (1)
4. Если (уровень соответствует заданному) и (его прирост положительный), то (кран надо медленно_закрывать) (1)
5. Если (уровень соответствует заданному) и (его прирост отрицательный), то (кран надо медленно_открывать) (1)

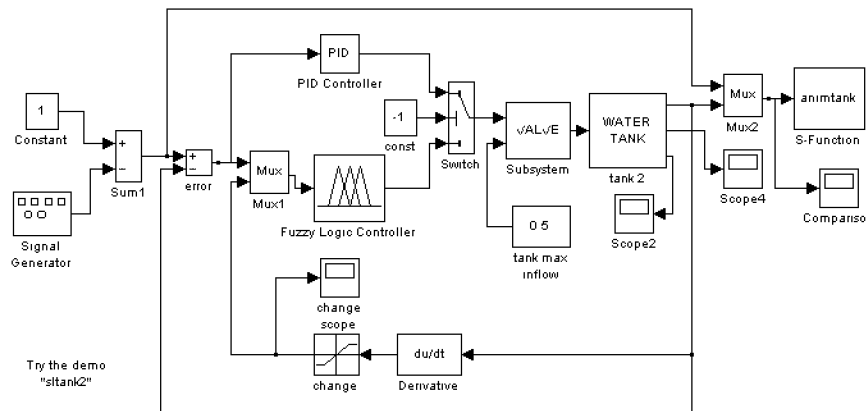


Рис. 5.45. Блок-диаграмма модели системы управления уровнем воды в баке с нечетким регулятором

Модель системы управления уровнем воды в баке с нечетким регулятором, основанная на приведенных правилах, является одной из демонстрационных моделей пакетов Fuzzy Logic и Simulink.

Ее можно вызвать из командной строки командой **sltank**, что приведет к открытию окна Simulink с блок-диаграммой указанной модели (рис. 5.45).

Одновременно блок Fuzzy Logic Controller будет сопоставлен с системой нечеткого вывода, записанной в файле tank.fis.

Для изучения процесса функционирования системы необходимо нажать кнопку **Start** панели инструментов блок-диаграммы модели и дважды щелкнуть левой кнопкой мыши по блоку Comparison (Сравнение). В появившемся окне этого блока будут показаны

изменяющиеся во времени сигналы заданного (последовательность импульсов прямоугольной формы) и фактического уровня воды (рис. 5.46). Как видно, переходный процесс в системе имеет аperiodическую форму и заканчивается достаточно быстро, т. е. качество регулирования следует признать хорошим.

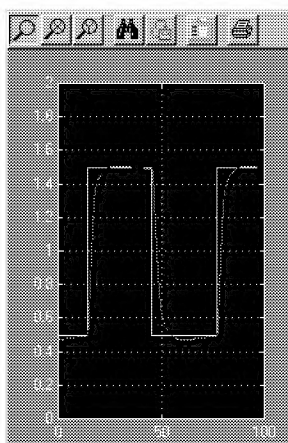


Рис. 5.46. Результаты моделирования системы управления с нечетким регулятором

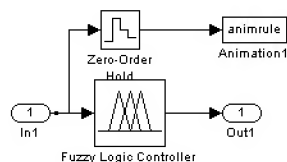


Рис. 5.47. Структура блока Fuzzy Logic Controller with Ruleviewer

Можно попытаться изменить характер функционирования системы, внося — например, с помощью рассмотренных программ с графическим интерфейсом (типа FIS-редактора и т. п.), определенные изменения в систему, задаваемую файлом tank.fis (изменяя правила, функции принадлежности, метод приведения к четкости и т. д.). Разумеется, все это лучше делать, остановив процесс моделирования.

Выполнив (в режиме командной строки) команду **sltankrule**, перейдем к блок диаграмме той же системы управления, но с нечетким регулятором с просмотрщиком правил (Fuzzy Logic Controller with Ruleviewer). Структура такого регулятора представляется в отдельном окне (рис. 5.47), если дважды щелкнуть левой кнопкой мыши по указанному блоку.

5.6.2. Построение нечеткой модели с использованием блоков Simulink. Для построения какой-то собственной моделирующей системы с использованием средств нечеткой логики и блоков Simulink рекомендуется просто скопировать блок Fuzzy Logic Controller из рассмотренной системы *sltank* (или какого-либо другого демонстрационного примера MATLAB) и поместить его в блок-диаграмму разрабатываемой системы. Из командной строки командой **fuzblock** можно также открыть библиотеку нечетких блоков пакета Simulink (в версии MATLAB 5.3, содержащей два блока: нечеткий регулятор — Fuzzy Logic Controller и нечеткий регулятор с просмотрщиком правил — Fuzzy Logic Controller with Ruleviewer, а в версии 5.2 — еще и несколько демонстрационных блоков), которые могут быть использованы точно так же, при необходимости — с дополнительным редактированием.

Отметим, что функционирование указанных блоков осуществляется с использованием системной S-функции **sffis.mex**. Запись этой функции такова:

output = sffis(t,x,u,flag,fismat),

где **output** — выход нечеткого регулятора, **t**, **x** и **flag** — стандартные аргументы системной функции Simulink, **fismat** — идентификатор (имя) нечеткой системы вывода, **u** — входной сигнал (вектор входных сигналов) регулятора.

По смыслу данная функция аналогична рассмотренной выше функции **evalfis**, но она оптимизирована для работы в среде Simulink.

5.6.3. Демонстрационные примеры работы с пакетом Fuzzy Logic Toolbox. Для ознакомления с пакетом Fuzzy Logic Toolbox можно использовать следующие функции (команды) в режиме командной строки:

defuzzdm — обзор методов приведения к четкости (дефазификации),

fcmdemo — демонстрация алгоритма кластеризации Fuzzy c-means (2-D графика),

fuzdemos — демонстрация графического интерфейса пакета Fuzzy Logic,

gasdemo — демонстрация использования аппарата гибридных сетей для решения задачи о выборе автомобиля, наиболее экономичного по расходу топлива,

juggler — демонстрация системы жонглирования мячом с использованием нечеткого регулятора,

invkine — демонстрация нечеткого управления движением робота-манипулятора,

irisfcm — демонстрация алгоритма кластеризации Fuzzy c-means,
noisedm — демонстрация решения задачи фильтрации на основе методов нечеткой логики,
slbb — демонстрация задачи «шар на качелях»,
slcp — демонстрация нечеткой системы управления перевернутым маятником,
sltank — демонстрация системы управления уровнем воды в баке с нечетким регулятором,
sltankrule — то же, что в предыдущем случае, но с дополнительным просмотром нечетких правил,
sltbu — демонстрация функционирования нечеткой системы управления грузовиком.

С каждым из этих примеров связан М-файл, fis-файл или/и блок-диаграмма Simulink, запуск которых производится с помощью одной из указанных команд. Текст пояснений в примерах — на английском языке. Данные примеры доступны и через главное меню MATLAB (пункт Help/Examples and Demos, раздел Toolboxes/Fuzzy Logic). Дополнительную информацию можно получить, используя команду **help fuzzy**.

СПИСОК ЛИТЕРАТУРЫ

1. Прикладные нечеткие системы / Под ред. Т. Тэрано, К. М.: Мир, 1993. 368 с.
2. Змитрович А.И. Интеллектуальные информационные системы / Минск: НТООО "ТетраСистемс", 1997. 367 с.
3. Уоссермен Ф. Нейрокомпьютерная техника.—М.: Мир, 1996. 275 с.
4. Горбань А.Н., Россиев Д.А. Нейронные сети на компьютере.—Новосибирск: Наука, 1996. 275 с.
5. Круглов В.В., Борисов В.В. Искусственные нейронные сети: теория и практика.—М.: Горячая линия—Телеком, 2001. 382 с.
6. Аверин А.Н., и др. Нечеткие множества в моделях управления и искусственного интеллекта / Под ред. Д.А.Поспелова.—М.: Наука, 1991. 271 с.
7. Кофман А., Алуха Х.Хил. Введение теории нечеткой логики в управление предприятием.—Минск: Высшая школа, 1992. 271 с.
8. Дли М.И. Локально-аппроксимационные модели сложных систем / Минск: НТООО "ТетраСистемс", 1997. 367 с.

СПИСОК ЛИТЕРАТУРЫ

1. Прикладные нечеткие системы / Под ред. Т. Тэрано, К. Асаи, М. Сугэно.— М.: Мир, 1993. 368 с.
2. Змитрович А.И. Интеллектуальные информационные системы.— Минск: НТООО "ТетраСистемс", 1997. 367 с.
3. Уоссермен Ф. Нейрокомпьютерная техника.—М.: Мир, 1992. 127 с.
4. Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере.—Новосибирск: Наука, 1996. 275 с.
5. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика.—М.: Горячая линия—Телеком, 2001. 382 с.
6. Аверин А.Н., и др. Нечеткие множества в моделях управления и искусственного интеллекта / Под ред. Д. А Пospelова.—М.: Наука, 1986. 312 с.
7. Кофман А., Алуха Х.Хил. Введение теории нечетких множеств в управление предприятием.—Минск: Высшая школа, 1992. 223 с.
8. Дли М.И. Локально-аппроксимационные модели сложных объектов.—М.: Наука, Физматлит, 1999. 112 с.
9. Дли М.И., Круглов В.В., Осокин М.В. Локально-аппроксимационные модели социально-экономических систем и процессов.—М.: Наука, Физматлит, 2000. 224 с.
10. Корнеев В.В., Греев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации.—М.: Нолидж, 2000. 352 с.